

Machine-Checked Formal Proofs (Lean 4): 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:** FormalProof_A45.lean (Lean 4.14.0 Kernel Verified)

Formal Verification Certificate (Lean 4 Interactive Theorem Prover)

The finite content of Article 45 — the agreement of the two frame actions, the classification of the twenty classes with their sizes and stabilisers, the descent of a letter transformation, the induced action on the classes, the worked instances of the variational criterion, and the printed table of Appendix A — has been formally specified and machine-checked in the **Lean 4** theorem prover (FormalProof_A45.lean, Section 3). **40 theorems** are proved.

- **Zero Unproven Placeholders (sorry-Free):** Every theorem is proved constructively down to the Lean 4 kernel by definitional equality (rfl) and, for the six arithmetic results that hold for every N , by induction and rewriting. No sorry, no axiom declaration, and no native_decide — the last would delegate evaluation to the compiler and would enter Lean.ofReduceBool in the audit below.
- **Zero External Dependencies:** Written strictly in Core Lean 4 (7-bit ASCII syntax) without Mathlib or any external library. Readers can verify the entire script at <https://live.lean-lang.org> or via lean FormalProof_A45.lean.
- **Kernel Axiom Audit (#print axioms):** As shown verbatim in Section 4, the Lean 4.14.0 kernel confirms that every one of the 40 theorems depends on **zero axioms** — not on propext, not on Classical.choice, and not on sorryAx. Four core-library lemmas, one library function and one tactic are replaced by local work for exactly this reason; see Section 2.
- **Scope Declared:** The file covers what Article 45 itself labels [A], finite or axiom-free. What is left out is named in Section 2, and the one substantial omission — the general- N orbit count, whose proof needs a theory of group actions — is partly recovered by two theorems proved for every N by induction rather than being passed over.

What a formalization adds over the computational dossier. The companion dossier for this article recomputes its quantities in Python and reports forty-five passing checks; agreement there is evidence, and two programs could in principle agree and both be wrong. A Lean proof is a different kind of object, and the difference is sharpest at two places in this article. Theorem 3.3 stops being a count that came out right and becomes a proof that two partitions of the sixty-four marker assignments are *equal*, membership checked both ways for every assignment. And the equivalence of the five conditions in Theorem 4.9 — the result whose first draft was false, because the condition now numbered (iv') is strictly stronger than the rest — is here a proof that five Boolean functions agree at every one of the twenty-four bijections, at three widths, with the failure of (iv') recorded as a theorem of its own rather than as a remark.

1 Index of Machine-Checked Theorems in FormalProof_A45.lean

Table 1: Correspondence between Lean 4 theorem identifiers in FormalProof_A45.lean and numbered results in Article 45.

Lean 4 Identifier	Manuscript Ref.	Tactic	Formally Verified Statement
frame_actions_agree	Lem. 2.15	rf1	The marker-to-word map W is a bijection $\{0, 1\}^{2N} \rightarrow \Sigma^N$ for $N = 1, \dots, 4$, and it intertwines the two frame actions: $W \circ \text{flip} = \rho^{\times N} \circ W$ and $W \circ \text{rev} = \text{reversal} \circ \tau^{\times N} \circ W$. The two frame groups of the article are therefore one group acting through W , which is what licenses passing between the marker picture and the word picture at all. Bijectivity is certified by the explicit two-sided inverse W_{inv} , checked both ways on every marker assignment and every word, rather than by counting the image.
witness_FFFBBF	Rem. 2.17	rf1	The witness carried over from the predecessor: FFFBBF has word (FF, FB, BF), with images (BB, BF, FB) and (FB, BF, FF) under the two generators.
orbit_count_enumerated	Lem. 3.1	rf1	The classes number 2, 6, 20, 72 at $2N = 2, 4, 6, 8$, agreeing in each case with $2^{2N-2} + 2^{N-1}$. Twenty at $2N = 6$ is the count the article is named for.
fixed_point_vector	Lem. 3.1, Proof 1	rf1	The fixed-point vector of the frame group on $\{0, 1\}^6$ is (64, 0, 8, 8), and Burnside averages it to twenty.
flip_is_fixed_point_free	Lem. 4.4(ii)	rf1	At four widths: flip is the unique element of Fr with no fixed point, the other two having 2^N each. This is what forces every orbit to have size 2 or 4 and never 1.
Dbar_separates_orbits	Thm. 3.3	rf1	The orbit map, as an equality of partitions. For every marker assignment, the set carrying its label has the size of its orbit <i>and</i> consists exactly of its orbit — membership checked in both directions over all sixty-four assignments. This is a proof that two partitions coincide, not a tally that agreed.
orbit_sizes_two_or_four	Prop. 3.6	rf1	No orbit has size one; every orbit has size 2 or 4; at $2N = 6$ there are eight of size 2 and twelve of size 4.
stabiliser_central_entry	Prop. 3.6	rf1	An orbit has size two exactly when its registry sequence is reversal-symmetric, and the two size-two families are then separated by the central entry D_N , which is 0 on the palindromic family and 1 on the anti-palindromic one.
type_counts_at_six	Cor. 3.7	rf1	The twenty classes split four, four and twelve by stabiliser type: the values 2^{N-1} , 2^{N-1} and $2^{2N-2} - 2^{N-1}$ at $N = 3$.
mulAssoc	—	induction	Associativity of multiplication on Nat, proved here because the core library's Nat.mul_assoc is itself proved through propext and importing it would have placed an axiom in the audit.
sq2	—	mulAssoc	$a \cdot 2 \cdot 2 = a \cdot 4$: doubling a doubled quantity quadruples it.
distrib4	—	Nat.mul_add	$a \cdot 2 + a \cdot 2 = a \cdot 4$.
powTwoStep	—	rw	$2^{2(N+1)} = 2^{2N} \cdot 4$: widening the stack by one site-pair multiplies the number of marker assignments by four. Nat.pow_add gives this in one line but is proved through propext, so the exponent is instead put into successor form by a definitional show.

continued on the next page

Table 1: (continued)

Lean 4 Identifier	Manuscript Ref.	Tactic	Formally Verified Statement
burnside_average	Lem. 3.1	rw	For every N, not at tabulated widths. The fixed-point vector $(2^{2N}, 0, 2^N, 2^N)$ averages over the four frame elements to exactly $2^{2N-2} + 2^{N-1}$. The arithmetic half of Lemma 3.1 in general; the passage from the vector to the orbit count is the group-theoretic half and is not formalized (Section 2).
orbit_decomposition	Cor. 3.7	rw	For every N. If f is the number of free orbits, so that $f + 2^N$ is the orbit count, then the 2^N orbits of size two and the f of size four account for every one of the 2^{2N} marker assignments and for no more. The count enters as a hypothesis so that no truncated subtraction occurs.
multiset_coincidence_is_isolated	Prop. 3.15(i)	rf1	Trap 4, first witness. At $N = 3$ the twenty classes and the twenty multisets agree in number; at $N = 4$ there are seventy-two classes against thirty-five multisets. The agreement is a coincidence of two counting functions at one point and carries no structure.
codon_profile_refutation	Prop. 3.15(ii)	rf1	Trap 4, second witness, in its corrected form. Counting alone does <i>not</i> refute the identification: $8 \cdot 2 + 12 \cdot 4 = 64$, so a surjection with the right total is arithmetically possible. What refutes it is the degeneracy profile — both profiles have twenty entries, the class profile is eight 2s and twelve 4s summing to 64, the code profile sums to 61 with 3 stops outside, and the two sorted profiles are unequal.
pinned_lists_are_the_derived_ones	Lem. 2.15, Thm. 3.3, Lem. 4.5, Prop. 5.5	rf1	The data are checked, not assumed. Nine enumerations are written into the file as data — the twenty registry classes, the word classifications at three widths, the twenty-four bijections, the eight admissible ones, and the registry classes at $N = 2, 3, 4$ — and this theorem certifies each by the property that characterises it: every entry is canonical, every element of the source has its image in the list, there are no repetitions, and the count is as stated. Recomputing each classification and comparing would re-derive it inside the comparison and does not terminate here; the property form also has no dependence on the order in which an enumeration happens to emit its elements.
eight_isomorphisms	Lem. 4.5	rf1	Exactly eight of the twenty-four bijections are isomorphisms of structured alphabets; four preserve each class of the partition and four exchange the two classes.
word_classes_match_marker_classes	Lem. 2.15, Cor. 3.7	rf1	The classification of words agrees with the classification of marker assignments, as Lemma 2.15 requires: 2, 6, 20 and 72 classes of words of length 1, 2, 3, 4, the third of them equal to the twenty marker classes. This is the theorem that detects a canonical representative chosen in the wrong base, which would split the twenty classes into twenty-two while disturbing no other count in the file.

continued on the next page

Table 1: (continued)

Lean 4 Identifier	Manuscript Ref.	Tactic	Formally Verified Statement
descent_conditions_equivalent	Thm. 4.9	rf1	The descent theorem, exhaustively. For every one of the twenty-four bijections and for $N = 1, 2, 3$, the five conditions (i)–(v) take the same truth value: every transformation, at three widths, every condition evaluated on every word.
exactly_eight_descend	Prop. 4.14	rf1	Exactly eight of the twenty-four bijections descend.
ivprime_strictly_stronger	Prop. 4.11	rf1	The corrected statement. The literal generator-matching condition (iv') is <i>strictly stronger</i> than the other four: for $N \geq 2$ it holds for precisely the four admissible transformations that preserve each class, so four of twenty-four against eight. The theorem as first drafted, with (iv') in place of (iv), would have been false, and this is the theorem that says so.
ivprime_witness	Prop. 4.11	rf1	The witness: the class-exchanging map (FF FB)(BB BF) is admissible and commutes with complementation, yet conjugates τ to $\tau\rho$ — which is exactly why intertwining holds for it and literal generator-matching does not.
descent_preserves_class_size	Thm. 4.9	rf1	Each of the eight admissible transformations induces a bijection of the classes preserving class size.
counterexample_meets_four_classes	Ex. 4.15	rf1	A transformation failing the condition does not distort the classification but destroys it: (BB BF) carries one class of size four onto a set of four words lying in four distinct classes, of sizes 2, 2, 4, 4.
kernel_is_id_and_rho	Prop. 5.3(i)	rf1	The kernel of the action on classes is $\{\text{id}, \rho\}$ at $N = 2, 3, 4$ and is the whole structural group at $N = 1$: the identity and complementation act trivially, and nothing else does.
psi_is_two_to_one	Prop. 5.3(iii)	rf1	The eight admissible transformations induce only <i>four</i> distinct maps of the twenty classes, each fibre of size two. No orbit-level quantity can separate more than four of them, which is the obstruction the whole of Section 5 is about.
kappa_lambda_well_defined	Prop. 5.5	rf1	κ and λ descend to the classes — each commutes with reversal up to reversal, so the image does not depend on the representative — and each carries classes to classes. Checked at $N = 2, 3, 4, 5$.
lambda_acts_freely	Prop. 5.5	rf1	Verified, not asserted. λ acts with no fixed class at $N = 2, 3, 4, 5$. The contrast is the point: $\text{Fix}(\kappa) = 2^N$ at every N , while $\kappa\lambda$ has fixed classes exactly when N is even — two at $N = 2$ and eight at $N = 4$, none at $N = 3$ or $N = 5$. An argument treating the three non-identity elements alike would be wrong, and an earlier draft did.
kappa_central_entry	Prop. 5.5	rf1	κ preserves the central entry on every reversal-symmetric class at every N checked, but on <i>every</i> class exactly when N is odd. The last two clauses are the counterexamples at $N = 2$ and $N = 4$, so the even case is a genuine failure and not an artefact of the representative.

continued on the next page

Table 1: (continued)

Lean 4 Identifier	Manuscript Ref.	Tactic	Formally Verified Statement
kappa_is_the_tau_action	Prop. 5.5(i)	rf1	The κ formula is not a definition made to fit: it <i>is</i> the letterwise action of τ transported through W . For each of the sixty-four assignments there is exactly one assignment whose word is τ applied letterwise, and its registry sequence is κ of the original.
admissible_commutes_with_rho	Thm. 5.8(i)	rf1	Every admissible transformation commutes with complementation, the conjugate of ρ being ρ . The residual indeterminacy is therefore exactly the coset of ρ : a two-element ambiguity and not a larger one.
phi_unique_maximiser	Ex. 6.14	rf1	A weight at which the four values of the functional are 1, 2, 4, 8 (that is $\frac{1}{2}, 1, 2, 4$ before clearing denominators), so the maximiser is unique and is $\kappa\lambda$.
phi_tie_from_symmetry	Ex. 6.15	rf1	A λ -invariant weight ties the four values in two pairs: on this point of the degeneracy locus the functional cannot distinguish a transformation from its composite with the class-inverting element.
Z_strictly_larger_than_Zmax	Rem. 6.17, Ex. 6.16	rf1	$Z \supsetneq Z_{\max}$, by witness . Two of the four values coincide, so the point lies on Z ; the maximiser is nevertheless unique; and neither the ensemble nor the weight carries a symmetry. Degeneracy of the functional is therefore not explained by symmetry of its data.
four_induced_maps_distinct	Prop. 6.11	rf1	The four induced maps are pairwise distinct on the twenty classes, so each of the six pair conditions cutting out Z is a nonzero bilinear form and Z is a proper subset of the parameter space.
lambda_has_ten_orbits	Rem. 6.18	rf1	$\langle \lambda \rangle$ has ten orbits, all of size two, on the twenty classes — the dimension the remark records for the space of tying weights.
tableA_agrees	App. A	rf1	The printed table, field by field. All twenty rows of Table A.1 — representative, label, size, stabiliser type, central entry, and the images under κ and λ — are recomputed from the definitions and agree. The table is also complete and without repetition: the twenty printed labels are distinct and every one of the twenty classes appears among them.
tableA_kappa_fixed	App. A	rf1	Exactly eight of the twenty classes carry the asterisk marking them κ -fixed, which is the 2^N of Proposition 5.5 at $N = 3$.

2 Scope of the Formalization

A formal certificate is informative only if its boundary is stated, and the boundary here is not arbitrary: it follows Article 45’s own status labels. What the article marks [A] — finite, or axiom-free arithmetic — is what a kernel can check, and the Lean file covers it. Four classes of claim lie outside, and are discharged in the manuscript by hypothesis, by citation, by numerical computation, or by a proof that needs a library this file does not load.

1. **Every statement about encoded stacks.** Everything proved below 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 H_{mk} , H_{stack} and H_{frm} , which the article registers and assumes. The Lean file neither assumes nor tests them.

2. **The general- N orbit count, as a count.** Lemma 3.1 holds for every N and is proved in the article three ways. What is formalized is the count by enumeration at $2N = 2, 4, 6, 8$, together with the Burnside *arithmetic* for every N (burnside_average) and the orbit decomposition for every N (orbit_decomposition). The remaining step — from a fixed-point vector to an orbit count — is the orbit-counting lemma itself, which needs a developed theory of group actions and hence Mathlib. This is the one place where the dossier proves less than the article, and it is said here rather than left to be inferred from the theorem list.
3. **The genetic-code datum.** The fibre profile of the standard table is an external empirical datum cited to [Cri68]. The file checks its internal arithmetic and its *inequality* with the profile of the twenty classes — which is the whole of the refutation — but not that the quoted table is the correct one.
4. **Anything resting on real numbers.** The entropy of Corollary 6.2 and the randomised word-level identity of Proposition 6.4 need logarithms and floating-point comparison; they are checked numerically in the computational dossier instead. The variational examples 6.14–6.16 *are* formalized: their data are dyadic, the ensemble is uniform on two classes, and clearing the denominator turns every comparison into an identity of natural numbers. Nothing is lost, since a maximiser of Φ is a maximiser of 2Φ .
5. **Two widths, for reasons of machine and not of mathematics.** The registry-class results are checked at $N = 2, 3, 4$ and the word classification at lengths 1, 2, 3; $N = 5$ and length 4 are not included. Nothing about them is in doubt — the article proves the general statements and the computational dossier checks $N = 5$ numerically — but the kernel reductions did not fit the memory of the machine this dossier was built on. What is lost is a second odd- N witness for the $\kappa\lambda$ dichotomy, which here rests on $N = 3$ against $N = 2$ and $N = 4$. This is recorded in the Lean file at the theorems concerned.
6. **Provenance and the status labels.** Which axioms a proposition invokes, and which label a statement carries, are properties of the text. They were established by reading and are recorded in the concordance and the defect ledger of Appendix B; no kernel can confirm them.

What could not be imported, and why. Article 62’s dossier reported one such substitution. `Nat.mul_assoc` is proved through `propext`, and `#print axioms` is transitive, so importing it would have made every downstream theorem report depends on `axioms: [propext]`. Auditing each candidate here before use turned up five more, and they are listed because the next dossier in this series will meet the same ones:

Core-library item	Reported by <code>#print axioms</code>
<code>Nat.mul_assoc</code>	<code>propext</code>
<code>Nat.pow_add</code>	<code>propext</code>
<code>Nat.add_mul</code>	<code>propext</code>
<code>List.mergeSort</code>	<code>propext</code> , <code>Quot.sound</code>
the omega tactic	<code>propext</code> , <code>Quot.sound</code>
<code>Nat.mul_comm</code> , <code>Nat.mul_add</code> , <code>Nat.pow_succ</code> , <code>Nat.add_assoc</code> , <code>Nat.add_comm</code> , <code>Nat.mul_succ</code> , <code>Nat.add_zero</code> , <code>Nat.two_mul</code>	<i>no axioms — used freely</i>

So associativity, the two distribution steps and the power identity are proved in the file; the two places that need a sorted list — the degeneracy profile of Proposition 3.15 and the class sizes of Example 4.15 — use an insertion sort defined there; duplicate removal is written out; and the arithmetic that omega would have discharged in two

lines is done by hand. None of this would have been a defect: `propext` and `Quot.sound` are two of Lean’s three standard axioms and are assumed throughout ordinary mathematics. But the stronger report is available for the cost of about forty lines, so it is taken.

Tamper test. An axiom audit shows that the theorems rest on nothing; it does not show that they *say* anything. A file of vacuous truths would audit just as cleanly. The formalization was therefore re-run against three perturbations of its defining data, each the kind of error a transcription would produce. Six data were perturbed, one at a time, each the kind of error a transcription would produce; each mutant was elaborated in full and every one exited 1. Complementation altered from the fixed-point-free involution to a four-cycle leaves **11** of the forty theorems unproved, and the canonicalising choice of representative reversed leaves **11**; the site-complementation map made trivial leaves **9**; the registry sequence read at the wrong offset leaves **5**; one row of Table A.1 altered in a single digit leaves **2**; and the reversal twist exchanged for the other transposition of the alphabet leaves **1**. Twenty-eight of the forty are broken by at least one perturbation.

Reading the spread, including where it is thin. The counts are reported individually rather than averaged, because what matters is that each perturbation breaks the checks that depend on the datum perturbed. The two large numbers are the two data everything rests on: complementation takes out the whole descent and induced-action cluster, the canonicalising choice takes out the classification and the variational examples, and both take out the pinning theorem, which is the check that the file’s data are the derived ones. The single break under the exchanged twist is not a blind spot but a result: the two candidate twists are carried to one another by a relabelling of the alphabet, so every statement internal to the word picture is invariant under the exchange, and the one theorem that is not — `frame_actions_agree` — is precisely the statement tying the letter-level twist to marker reversal through *W*. The article makes the same point at Lemma 4.4(i).

What the test does not reach. Twelve theorems are left standing by every perturbation. Six of them — `mulAssoc`, `sq2`, `distrib4`, `powTwoStep`, `burnside_average` and `orbit_decomposition` — are arithmetic over `Nat` and depend on none of the perturbed data by construction, so their insensitivity is correct rather than suspicious. The remaining six — `orbit_count_enumerated`, `multiset_coincidence_is_isolated`, `eight_isomorphisms`, `word_classes_match_marker_classes`, `kappa_central_entry` and `four_induced_maps_distinct` — turn on data this test did not perturb, chiefly the registry partition of the alphabet and the enumeration of marker assignments. They are covered by no mutant here, and that is stated rather than left for a reader to work out from the lists. The spread is reported rather than averaged, because what matters is that each perturbation breaks the checks that depend on the datum perturbed, and not how many it breaks: the registry sequence enters nearly every result, the letter encoding of the twist enters the word picture and the induced action, and a single table row enters only the two theorems about the table.

On `sorryAx`. Lean does not refuse a file whose tactic blocks fail; it reports the error, admits the theorem, and marks it as depending on `sorryAx`. A dossier that printed its theorem list without printing the audit could therefore display a page of green theorems that the kernel considers unproved. That is why Section 4 reproduces the compiler’s own `stdout` rather than a summary of it, and why the exit status is quoted.

3 Complete Lean 4 Formal Proof Script (`FormalProof_A45.lean`)

Browser & CLI Verification: Copy the listing below directly into the official Lean 4 web prover at <https://live.lean-lang.org> (or save it as `FormalProof_A45.lean` and run `lean FormalProof_A45.lean`). It needs no project, no lakefile, and no network access. The `set_option maxRecDepth` on the first line is required: the kernel reduces enumerations over $\{0, 1\}^9$ and over all twenty-four letter bijections during elaboration, and the default depth is not sufficient for them.

```

1  /-
2  FormalProof_A45.lean
3  Machine-Checked Formal Proof Dossier for Article 45:
4  "The Twenty Registry Classes of a Chrysene Stack: Frame Orbits, Descent of
5  a Letter Transformation, and a Variational Criterion"
6  Annals of the Chrysene Formalism, Vol. 1, No. 2, pp. 676-704 (2026).
7  DOI: https://doi.org/10.68366/vfjz1403
8
9  Verified in Lean 4 (Core library only; pure ASCII, zero `sorry`, zero
10 external dependencies, no `native_decide`).
11
12 SCOPE. This file formalizes the FINITE content of Article 45 -- those
13 statements the article itself labels [A], decidable by enumeration over a
14 finite set or by arithmetic over stated integers. It does not formalize,
15 and does not attempt to:
16
17 * any statement about encoded stacks as opposed to binary strings. The
18 passage from  $\{0,1\}^{2N}$  to an encoded chrysene stack is the content of
19 the realization hypotheses  $H_{mk}$ ,  $H_{stack}$  and  $H_{frm}$ , which the article
20 registers and assumes. Everything below is a statement about binary
21 strings and about words over a four-letter alphabet.
22 * the general- $N$  orbit count as such. Lemma 3.1 is proved in the article
23 by Burnside for every  $N$ . What is proved below is the count by
24 enumeration at  $2N = 2, 4, 6, 8$ , together with the Burnside ARITHMETIC
25 for every  $N$  by induction (`burnside_average`) and the orbit
26 decomposition for every  $N$  (`orbit_decomposition`). The step from a
27 fixed-point vector to an orbit count is the group-theoretic content
28 and needs a developed theory of group actions, hence Mathlib.
29 * the genetic-code datum. The fibre profile of the standard table is an
30 external empirical datum cited to [Cri68]. What is checked below is
31 its internal arithmetic and its INEQUALITY with the profile of the
32 twenty classes, which is the whole of the refutation in Prop. 3.15.
33 * anything resting on real numbers: the entropy of Corollary 6.2 and the
34 randomised word-level identity of Proposition 6.4. These are checked
35 numerically in the computational dossier, verify_twenty_classes.py.
36 The variational examples 6.14-6.16 ARE formalized, in integer form:
37 their data are dyadic and are cleared of denominators below.
38 * the provenance claims. Which axioms a proposition invokes and which
39 status label it carries are properties of the text, not mathematical
40 statements; they are recorded in the concordance of Appendix B.
41
42 The division is not a convenience. It follows the article's own status
43 labels: what is marked [A] is what a kernel can check, and the two agree.
44
45 CONVENTIONS. Markers are lists over  $\{0,1\}$  of length  $2N$ , in the offset-0
46 convention of Section 2. Letters are encoded  $FF = 0$ ,  $FB = 1$ ,  $BF = 2$ ,
47  $BB = 3$ ; the registry partition is  $\{\{FF,BB\},\{FB,BF\}\} = \{\{0,3\},\{1,2\}\}$ ;  $\rho$ 
48 (complementation) is  $x \mapsto 3 - x$  and  $\tau$  is the transposition  $(FB\ BF)$ .
49 These are the encodings used by the computational dossier, so that the two
50 artifacts can be compared line by line.
51
52 ON IMPORTS. Three core-library facilities that would have shortened this
53 file are not used, because `#print axioms` is transitive and each would
54 have put an axiom in the audit: `Nat.mul_assoc` is proved through
55 `propext`, and `List.mergeSort` through `propext` AND `Quot.sound`.
56 Associativity is proved below by induction, and the two places that need a
57 sorted list use the insertion sort defined below.
58

```

```

59  ON KERNEL COST. Every enumeration used more than once is bound to a
60  top-level constant (`B6`, `X5`, `CW3`, `P24`, ...) rather than written out
61  at each use. This is not cosmetic. The kernel caches reduction by
62  expression, so a repeated constant is computed once; an inline
63  `Xcl 5` appearing inside a quantifier over `Xcl 5` is instead recomputed
64  once per element, which for the 272 classes at N = 5 is the difference
65  between a file that checks and a file that exhausts memory. For the same
66  reason `nub` is written with an accumulator: the natural recursion binds
67  its tail twice, the kernel substitutes rather than shares, and reduction
68  becomes exponential in the length of the list.
69  -/
70
71  set_option maxRecDepth 100000
72  -- The enumerations below exceed the default elaboration budget; the
73  -- outer `timeout` of the build, not a heartbeat count, is what bounds
74  -- this file.
75  set_option maxHeartbeats 0
76
77  namespace ChryseneFormalism.Article45
78
79  -- =====
80  -- 0. Finite enumeration helpers (Core Lean only)
81  -- =====
82
83  /-- All lists of length `n` over the given alphabet. -/
84  def allOver (alph : List Nat) : Nat -> List (List Nat)
85  | 0 => [[]]
86  | n + 1 => ((allOver alph n).map (fun t => alph.map (fun a => a :: t))).flatten
87
88  /-- All binary strings of length `n`: the marker assignments of n sites. -/
89  def allBits (n : Nat) : List (List Nat) := allOver [0, 1] n
90
91  /-- All words of length `n` over the four-letter alphabet. -/
92  def allWords (n : Nat) : List (List Nat) := allOver [0, 1, 2, 3] n
93
94  /-- Duplicate removal, written here rather than imported. The accumulator
95  is not cosmetic: the natural recursion `let r := nub t; if r.contains a
96  then r else a :: r` uses `r` twice, the kernel substitutes the binding
97  rather than sharing it, and reduction of a sixty-four element list then
98  takes exponentially many steps. This form is linear in the kernel as
99  well as in the compiler. -/
100 def nubAux : List (List Nat) -> List (List Nat) -> List (List Nat)
101 | acc, [] => acc.reverse
102 | acc, a :: t => if acc.contains a then nubAux acc t else nubAux (a :: acc) t
103
104 def nub (l : List (List Nat)) : List (List Nat) := nubAux [] l
105
106 /-- Insertion sort on naturals, replacing `List.mergeSort`, which is proved
107 through `propext` and `Quot.sound` and would appear in the audit. -/
108 def insNat (x : Nat) : List Nat -> List Nat
109 | [] => [x]
110 | a :: t => if Nat.blt x a then x :: a :: t else a :: insNat x t
111
112 def sortNat : List Nat -> List Nat
113 | [] => []
114 | a :: t => insNat a (sortNat t)
115
116 def countP (p : List Nat -> Bool) (l : List (List Nat)) : Nat := (l.filter p).length

```

```

117
118 /-- The marker enumerations used below, bound once. -/
119 def B2 : List (List Nat) := allBits 2
120 def B4 : List (List Nat) := allBits 4
121 def B6 : List (List Nat) := allBits 6
122 def B8 : List (List Nat) := allBits 8
123
124 /-- The registry-sequence enumerations: lists of length 2N-1 for
125     N = 2, 3, 4, 5. -/
126 def S3 : List (List Nat) := allBits 3
127 def S5 : List (List Nat) := allBits 5
128 def S7 : List (List Nat) := allBits 7
129
130 /-- The word enumerations for N = 1, 2, 3, 4. Wd4 is needed for the
131     twenty-four letter bijections and for the multiset count at N = 4; the
132     classification of words of length four is not formed. -/
133 def Wd1 : List (List Nat) := allWords 1
134 def Wd2 : List (List Nat) := allWords 2
135 def Wd3 : List (List Nat) := allWords 3
136 def Wd4 : List (List Nat) := allWords 4
137
138 -----
139 -- Pinned enumerations.
140 --
141 -- The classifications below are given as data and then CHECKED against
142 -- the definitions by `pinned_lists_are_the_derived_ones`. They are not
143 -- assumptions: that theorem recomputes each one from scratch and compares.
144 --
145 -- They are pinned because a derived list behaves badly under kernel
146 -- reduction. `nub (xs.map f)` holds unevaluated applications of `f`, and
147 -- every membership test re-reduces them, so a quantifier over a derived
148 -- classification re-derives it once per element. Pinning turns that into
149 -- a traversal of given data and is the difference between a file that
150 -- checks in minutes and one that exhausts twelve gigabytes.
151 -----
152
153 /-- the twenty registry classes at 2N = 6 (20 entries), pinned as data; the theorem
154     `pinned_lists_are_the_derived_ones` recomputes this from the
155     definitions and compares, so it is checked and not assumed. -/
156 def C20L : List (List Nat) :=
157   [[0, 0, 0, 0, 0],
158    [0, 0, 0, 0, 1],
159    [0, 0, 0, 1, 1],
160    [0, 0, 0, 1, 0],
161    [0, 0, 1, 1, 0],
162    [0, 0, 1, 1, 1],
163    [0, 0, 1, 0, 1],
164    [0, 0, 1, 0, 0],
165    [0, 1, 1, 0, 1],
166    [0, 1, 1, 1, 1],
167    [0, 1, 1, 1, 0],
168    [0, 1, 0, 1, 0],
169    [0, 1, 0, 1, 1],
170    [0, 1, 0, 0, 1],
171    [1, 0, 0, 1, 1],
172    [1, 1, 0, 1, 1],
173    [1, 1, 1, 1, 1],
174    [1, 0, 1, 1, 1],

```

```

175 [1, 0, 1, 0, 1],
176 [1, 0, 0, 0, 1]]
177
178 /-- the classes of words of length one (2 entries), pinned as data; the theorem
179   `pinned_lists_are_the_derived_ones` recomputes this from the
180   definitions and compares, so it is checked and not assumed. -/
181 def CW1L : List (List Nat) :=
182   [[0],
183    [1]]
184
185 /-- the classes of words of length two (6 entries), pinned as data; the theorem
186   `pinned_lists_are_the_derived_ones` recomputes this from the
187   definitions and compares, so it is checked and not assumed. -/
188 def CW2L : List (List Nat) :=
189   [[0, 0],
190    [0, 2],
191    [0, 1],
192    [0, 3],
193    [1, 1],
194    [1, 2]]
195
196 /-- the twenty classes of words of length three (20 entries), pinned as data; the theorem
197   `pinned_lists_are_the_derived_ones` recomputes this from the
198   definitions and compares, so it is checked and not assumed. -/
199 def CW3L : List (List Nat) :=
200   [[0, 0, 0],
201    [0, 0, 2],
202    [0, 0, 1],
203    [0, 0, 3],
204    [0, 1, 0],
205    [0, 2, 2],
206    [0, 2, 1],
207    [0, 2, 3],
208    [0, 1, 2],
209    [0, 1, 1],
210    [0, 1, 3],
211    [0, 3, 0],
212    [0, 3, 2],
213    [0, 3, 1],
214    [1, 0, 1],
215    [1, 3, 2],
216    [1, 1, 1],
217    [1, 1, 2],
218    [1, 2, 1],
219    [1, 0, 2]]
220
221 /-- the twenty-four bijections of the alphabet (24 entries), pinned as data; the theorem
222   `pinned_lists_are_the_derived_ones` recomputes this from the
223   definitions and compares, so it is checked and not assumed. -/
224 def P24L : List (List Nat) :=
225   [[3, 2, 1, 0],
226    [2, 3, 1, 0],
227    [3, 1, 2, 0],
228    [1, 3, 2, 0],
229    [2, 1, 3, 0],
230    [1, 2, 3, 0],
231    [3, 2, 0, 1],
232    [2, 3, 0, 1],

```

```

233 [3, 0, 2, 1],
234 [0, 3, 2, 1],
235 [2, 0, 3, 1],
236 [0, 2, 3, 1],
237 [3, 1, 0, 2],
238 [1, 3, 0, 2],
239 [3, 0, 1, 2],
240 [0, 3, 1, 2],
241 [1, 0, 3, 2],
242 [0, 1, 3, 2],
243 [2, 1, 0, 3],
244 [1, 2, 0, 3],
245 [2, 0, 1, 3],
246 [0, 2, 1, 3],
247 [1, 0, 2, 3],
248 [0, 1, 2, 3]]
249
250 /-- the eight admissible bijections (8 entries), pinned as data; the theorem
251   `pinned_lists_are_the_derived_ones` recomputes this from the
252   definitions and compares, so it is checked and not assumed. -/
253 def ADML : List (List Nat) :=
254   [[3, 2, 1, 0],
255    [3, 1, 2, 0],
256    [2, 3, 0, 1],
257    [2, 0, 3, 1],
258    [1, 3, 0, 2],
259    [1, 0, 3, 2],
260    [0, 2, 1, 3],
261    [0, 1, 2, 3]]
262
263 /-- the registry classes at N = 2 (6 entries), pinned as data; the theorem
264   `pinned_lists_are_the_derived_ones` recomputes this from the
265   definitions and compares, so it is checked and not assumed. -/
266 def X2L : List (List Nat) :=
267   [[0, 0, 0],
268    [0, 0, 1],
269    [0, 1, 0],
270    [0, 1, 1],
271    [1, 0, 1],
272    [1, 1, 1]]
273
274 /-- the registry classes at N = 3 (20 entries), pinned as data; the theorem
275   `pinned_lists_are_the_derived_ones` recomputes this from the
276   definitions and compares, so it is checked and not assumed. -/
277 def X3L : List (List Nat) :=
278   [[0, 0, 0, 0, 0],
279    [0, 0, 0, 0, 1],
280    [0, 0, 0, 1, 0],
281    [0, 0, 0, 1, 1],
282    [0, 0, 1, 0, 0],
283    [0, 0, 1, 0, 1],
284    [0, 0, 1, 1, 0],
285    [0, 0, 1, 1, 1],
286    [0, 1, 0, 0, 1],
287    [0, 1, 0, 1, 0],
288    [0, 1, 0, 1, 1],
289    [0, 1, 1, 0, 1],
290    [0, 1, 1, 1, 0],

```

```

291 [0, 1, 1, 1, 1],
292 [1, 0, 0, 0, 1],
293 [1, 0, 0, 1, 1],
294 [1, 0, 1, 0, 1],
295 [1, 0, 1, 1, 1],
296 [1, 1, 0, 1, 1],
297 [1, 1, 1, 1, 1]]
298
299 /-- the registry classes at N = 4 (72 entries), pinned as data; the theorem
300   `pinned_lists_are_the_derived_ones` recomputes this from the
301   definitions and compares, so it is checked and not assumed. -/
302 def X4L : List (List Nat) :=
303   [[0, 0, 0, 0, 0, 0, 0],
304    [0, 0, 0, 0, 0, 0, 1],
305    [0, 0, 0, 0, 0, 1, 0],
306    [0, 0, 0, 0, 0, 1, 1],
307    [0, 0, 0, 0, 1, 0, 0],
308    [0, 0, 0, 0, 1, 0, 1],
309    [0, 0, 0, 0, 1, 1, 0],
310    [0, 0, 0, 0, 1, 1, 1],
311    [0, 0, 0, 1, 0, 0, 0],
312    [0, 0, 0, 1, 0, 0, 1],
313    [0, 0, 0, 1, 0, 1, 0],
314    [0, 0, 0, 1, 0, 1, 1],
315    [0, 0, 0, 1, 1, 0, 0],
316    [0, 0, 0, 1, 1, 0, 1],
317    [0, 0, 0, 1, 1, 1, 0],
318    [0, 0, 0, 1, 1, 1, 1],
319    [0, 0, 1, 0, 0, 0, 1],
320    [0, 0, 1, 0, 0, 1, 0],
321    [0, 0, 1, 0, 0, 1, 1],
322    [0, 0, 1, 0, 1, 0, 0],
323    [0, 0, 1, 0, 1, 0, 1],
324    [0, 0, 1, 0, 1, 1, 0],
325    [0, 0, 1, 0, 1, 1, 1],
326    [0, 0, 1, 1, 0, 0, 1],
327    [0, 0, 1, 1, 0, 1, 0],
328    [0, 0, 1, 1, 0, 1, 1],
329    [0, 0, 1, 1, 1, 0, 0],
330    [0, 0, 1, 1, 1, 0, 1],
331    [0, 0, 1, 1, 1, 1, 0],
332    [0, 0, 1, 1, 1, 1, 1],
333    [0, 1, 0, 0, 0, 0, 1],
334    [0, 1, 0, 0, 0, 1, 0],
335    [0, 1, 0, 0, 0, 1, 1],
336    [0, 1, 0, 0, 1, 0, 1],
337    [0, 1, 0, 0, 1, 1, 0],
338    [0, 1, 0, 0, 1, 1, 1],
339    [0, 1, 0, 1, 0, 0, 1],
340    [0, 1, 0, 1, 0, 1, 0],
341    [0, 1, 0, 1, 0, 1, 1],
342    [0, 1, 0, 1, 1, 0, 1],
343    [0, 1, 0, 1, 1, 1, 0],
344    [0, 1, 0, 1, 1, 1, 1],
345    [0, 1, 1, 0, 0, 0, 1],
346    [0, 1, 1, 0, 0, 1, 1],
347    [0, 1, 1, 0, 1, 0, 1],
348    [0, 1, 1, 0, 1, 1, 0],

```

```

349 [0, 1, 1, 0, 1, 1, 1],
350 [0, 1, 1, 1, 0, 0, 1],
351 [0, 1, 1, 1, 0, 1, 1],
352 [0, 1, 1, 1, 1, 0, 1],
353 [0, 1, 1, 1, 1, 1, 0],
354 [0, 1, 1, 1, 1, 1, 1],
355 [1, 0, 0, 0, 0, 0, 1],
356 [1, 0, 0, 0, 0, 1, 1],
357 [1, 0, 0, 0, 1, 0, 1],
358 [1, 0, 0, 0, 1, 1, 1],
359 [1, 0, 0, 1, 0, 0, 1],
360 [1, 0, 0, 1, 0, 1, 1],
361 [1, 0, 0, 1, 1, 0, 1],
362 [1, 0, 0, 1, 1, 1, 1],
363 [1, 0, 1, 0, 0, 1, 1],
364 [1, 0, 1, 0, 1, 0, 1],
365 [1, 0, 1, 0, 1, 1, 1],
366 [1, 0, 1, 1, 0, 1, 1],
367 [1, 0, 1, 1, 1, 0, 1],
368 [1, 0, 1, 1, 1, 1, 1],
369 [1, 1, 0, 0, 0, 1, 1],
370 [1, 1, 0, 0, 1, 1, 1],
371 [1, 1, 0, 1, 0, 1, 1],
372 [1, 1, 0, 1, 1, 1, 1],
373 [1, 1, 1, 0, 1, 1, 1],
374 [1, 1, 1, 1, 1, 1, 1]]
375
376 -- =====
377 -- 1. Markers, the frame group, and the registry sequence
378 -- (Definitions 2.9 and 2.11)
379 -- =====
380
381 /-- Complementation of every site: the fixed-point-free frame element. -/
382 def flipM (m : List Nat) : List Nat := m.map (fun x => 1 - x)
383
384 /-- Reversal of the site order. -/
385 def revM (m : List Nat) : List Nat := m.reverse
386
387 /-- Definition 2.11. The registry sequence  $D_n = m_n + m_{n+1}$  in  $\mathbb{Z}/2$ , of
388 length  $2N - 1$ , hence of odd length with a well-defined central entry. -/
389 def Dseq (m : List Nat) : List Nat :=
390   (List.range (m.length - 1)).map (fun i => (m.getD i 0 + m.getD (i + 1) 0) % 2)
391
392 /-- A list of digits read as a base-four numeral. It is used only to
393 order equal-length lists, where the numerical and the lexicographic
394 orders agree, so that "the least element" names a canonical
395 representative. The base must be four and not two: words are over a
396 four-letter alphabet, and in base two the distinct words (BF, FF) and
397 (FB, FF) both read as 4, so the least element of an orbit would not be
398 well defined and the twenty classes would be counted as twenty-two. -/
399 def toNat (d : List Nat) : Nat := d.foldl (fun a b => 4 * a + b) 0
400
401 /-- Definition 2.11. The registry class: the sequence modulo reversal. -/
402 def Dbar (d : List Nat) : List Nat :=
403   if Nat.ble (toNat d) (toNat d.reverse) then d else d.reverse
404
405 /-- The class label of a marker assignment. -/
406 def clsM (m : List Nat) : List Nat := Dbar (Dseq m)

```

```

407
408 /-- The frame orbit of a marker assignment, as a duplicate-free list. -/
409 def orbitM (m : List Nat) : List (List Nat) :=
410   nub [m, flipM m, revM m, flipM (revM m)]
411
412 /-- The classes at a given width. -/
413 def classesAt (twoN : Nat) : List (List Nat) := nub ((allBits twoN).map clsM)
414
415 /-- The twenty classes at 2N = 6, as pinned above. -/
416 abbrev C20 : List (List Nat) := C20L
417
418 /-- Stabiliser type: 1 trivial, 2 = {id, rev} (palindromic),
419   3 = {id, flip rev} (anti-palindromic). -/
420 def stabType (m : List Nat) : Nat :=
421   if revM m == m then 2 else if flipM (revM m) == m then 3 else 1
422
423 -- =====
424 -- 2. Lemma 2.15: the two frame actions agree
425 -- =====
426
427 /-- The offset-0 marker-to-word map W of Lemma 2.15. The i-th letter is
428   determined by the registry bit b1 = m_{2i} + m_{2i+1} and the face bit
429   b2 = m_{2i+1}: (0,0) |-> FF, (0,1) |-> BB, (1,0) |-> BF, (1,1) |-> FB. -/
430 def W (m : List Nat) : List Nat :=
431   (List.range (m.length / 2)).map (fun i =>
432     let b1 := (m.getD (2 * i) 0 + m.getD (2 * i + 1) 0) % 2
433     let b2 := m.getD (2 * i + 1) 0 % 2
434     if b1 == 0 then (if b2 == 0 then 0 else 3) else (if b2 == 0 then 2 else 1))
435
436 /-- Complementation on letters: FF <-> BB and FB <-> BF. -/
437 def rhoL (x : Nat) : Nat := 3 - x
438
439 /-- The reversal twist on letters: FB <-> BF, fixing the aligned class. -/
440 def tauL (x : Nat) : Nat := if x == 1 then 2 else if x == 2 then 1 else x
441
442 def flipW (w : List Nat) : List Nat := w.map rhoL
443 def revW (w : List Nat) : List Nat := (w.map tauL).reverse
444
445 /-- The inverse of W, read off its definition: the letter determines the
446   face bit b2 = m_{2i+1} directly, and then m_{2i} = b1 + b2. -/
447 def Winv (w : List Nat) : List Nat :=
448   (w.map (fun x =>
449     if x == 0 then [0, 0] else if x == 1 then [0, 1]
450     else if x == 2 then [1, 0] else [1, 1])).flatten
451
452 /-- W is a bijection of the given marker list onto the given word list, and
453   intertwines the two frame actions on it. Bijectivity is certified by
454   exhibiting a two-sided inverse rather than by counting the image: the
455   inverse is checked in both directions on every element, which is linear,
456   whereas duplicate-removal over the image is quadratic and at 2N = 8 does
457   not finish in reasonable memory. -/
458 def frameOK (ms ws : List (List Nat)) : Bool :=
459   (ms.all (fun m => Winv (W m) == m)) &&
460   (ws.all (fun w => W (Winv w) == w)) &&
461   (ms.all (fun m => (W (flipM m) == flipW (W m)) && (W (revM m) == revW (W m))))
462
463 /-- Lemma 2.15. W is a bijection of {0,1}^{2N} onto Sigma^N for
464   N = 1, 2, 3, 4, and it intertwines the marker frame action with the word

```

```

465   frame action: W o flip = rho^N o W and W o rev = reversal o tau^N o W.
466   Bijectivity is witnessed by the explicit inverse Winv, checked both
467   ways on every marker assignment and every word.
468   The two frame groups of the article are therefore one group acting
469   through W, which is what licenses passing between the two pictures. -/
470 theorem frame_actions_agree :
471   (frameOK B2 Wd1 && frameOK B4 Wd2 && frameOK B6 Wd3 && frameOK B8 Wd4) = true := by
472   rfl
473
474 /-- Remark 2.17. The witness carried over from the predecessor: the marker
475   assignment FFFBBF has word (FF, FB, BF), and its images under the two
476   frame generators are (BB, BF, FB) and (FB, BF, FF). -/
477 theorem witness_FFFBBF :
478   W [0, 0, 0, 1, 1, 0] = [0, 1, 2] /\
479   W (flipM [0, 0, 0, 1, 1, 0]) = [3, 2, 1] /\
480   W (revM [0, 0, 0, 1, 1, 0]) = [1, 2, 0] := by
481   repeat (first | rfl | constructor)
482
483 -- =====
484 -- 3. Lemma 3.1, Theorem 3.3, Proposition 3.6: the classification
485 -- =====
486
487 /-- Lemma 3.1, by enumeration. The classes number 2, 6, 20, 72 at
488   2N = 2, 4, 6, 8, agreeing in each case with 2^{2N-2} + 2^{N-1}. Twenty
489   is the count at 2N = 6, which is the title of the article. -/
490 theorem orbit_count_enumerated :
491   (classesAt 2).length = 2 /\
492   (classesAt 4).length = 6 /\
493   C20.length = 20 /\
494   (classesAt 8).length = 72 /\
495   (classesAt 2).length = 2 ^ 0 + 2 ^ 0 /\
496   (classesAt 4).length = 2 ^ 2 + 2 ^ 1 /\
497   C20.length = 2 ^ 4 + 2 ^ 2 /\
498   (classesAt 8).length = 2 ^ 6 + 2 ^ 3 := by
499   repeat (first | rfl | constructor)
500
501 /-- Lemma 3.1, Proof 1. The fixed-point vector of the frame group on
502   {0,1}^6 is (64, 0, 8, 8), and Burnside averages it to twenty. -/
503 theorem fixed_point_vector :
504   countP (fun m => m == m) B6 = 64 /\
505   countP (fun m => flipM m == m) B6 = 0 /\
506   countP (fun m => revM m == m) B6 = 8 /\
507   countP (fun m => flipM (revM m) == m) B6 = 8 /\
508   (64 + 0 + 8 + 8) / 4 = 20 := by
509   repeat (first | rfl | constructor)
510
511 /-- flip has no fixed point on the given marker list, while the other two
512   non-identity frame elements have `k` each. -/
513 def fpfOK (ms : List (List Nat)) (k : Nat) : Bool :=
514   (countP (fun m => flipM m == m) ms == 0) &&
515   (countP (fun m => revM m == m) ms == k) &&
516   (countP (fun m => flipM (revM m) == m) ms == k)
517
518 /-- Lemma 4.4(ii) in marker coordinates, at four widths: flip is the unique
519   element of the frame group with no fixed point, the other two having
520   2^N each. This is what forces every orbit to have size 2 or 4. -/
521 theorem flip_is_fixed_point_free :
522   (fpfOK B2 2 && fpfOK B4 4 && fpfOK B6 8 && fpfOK B8 16) = true := by rfl

```

```

523
524 /-- Theorem 3.3. The orbit map IS the registry sequence modulo reversal.
525 Stated as an equality of two partitions of the sixty-four marker
526 assignments: for every assignment, the set carrying its label has the
527 size of its orbit, and consists exactly of the members of its orbit.
528 This is not a count that agreed; it is a check of membership in both
529 directions, over every assignment. -/
530 theorem Dbar_separates_orbits :
531   (B6.all (fun m =>
532     (countP (fun x => clsM x == clsM m) B6 == (orbitM m).length) &&
533     (B6.all (fun x => (clsM x == clsM m) == (orbitM m).contains x))))
534   = true := by rfl
535
536 /-- Proposition 3.6. No orbit has size one; every orbit has size 2 or 4;
537 and at 2N = 6 there are eight of size 2 and twelve of size 4. -/
538 theorem orbit_sizes_two_or_four :
539   (B6.all (fun m => ((orbitM m).length == 2) || ((orbitM m).length == 4))) = true /\
540   (B6.all (fun m => !((orbitM m).length == 1))) = true /\
541   countP (fun c => countP (fun m => clsM m == c) B6 == 2) C20 = 8 /\
542   countP (fun c => countP (fun m => clsM m == c) B6 == 4) C20 = 12 := by
543   repeat (first | rfl | constructor)
544
545 /-- Proposition 3.6, the stabiliser criterion. An orbit has size two exactly
546 when its registry sequence is reversal-symmetric; the two families of
547 size-two orbits are separated by the CENTRAL ENTRY of that sequence,
548 which is 0 on the palindromic family and 1 on the anti-palindromic one.
549 Checked over all sixty-four assignments. -/
550 theorem stabiliser_central_entry :
551   (B6.all (fun m =>
552     (((orbitM m).length == 2) == (Dseq m == (Dseq m).reverse)) &&
553     (if stabType m == 2 then (Dseq m).getD 2 0 == 0 else true) &&
554     (if stabType m == 3 then (Dseq m).getD 2 0 == 1 else true) &&
555     ((stabType m == 1) == ((orbitM m).length == 4))))
556   = true := by rfl
557
558 /-- Corollary 3.7. At 2N = 6 the twenty classes split four palindromic,
559 four anti-palindromic and twelve free: the values  $2^{N-1}$ ,  $2^{N-1}$  and
560  $2^{2N-2} - 2^{N-1}$  at N = 3. -/
561 theorem type_counts_at_six :
562   countP (fun c => !(countP (fun m => (clsM m == c) && (stabType m == 2)) B6 == 0))
563     C20 = 4 /\
564   countP (fun c => !(countP (fun m => (clsM m == c) && (stabType m == 3)) B6 == 0))
565     C20 = 4 /\
566   countP (fun c => !(countP (fun m => (clsM m == c) && (stabType m == 1)) B6 == 0))
567     C20 = 12 /\
568   4 = 2 ^ 2 /\ 12 = 2 ^ 4 - 2 ^ 2 := by
569   repeat (first | rfl | constructor)
570
571 -- =====
572 -- 4. Two general-N identities, proved by induction for every N
573 -- =====
574
575 /-- Associativity of multiplication on Nat, proved here by induction rather
576 than imported: `Nat.mul_assoc` is proved through `propext`, and
577 importing it would have placed an axiom in the audit below. -/
578 theorem mulAssoc : forall a b c : Nat, a * b * c = a * (b * c)
579 | _, _, 0 => rfl
580 | a, b, (c + 1) => by

```

```

581   show a * b * c + a * b = a * (b * c + b)
582   rw [Nat.mul_add, mulAssoc a b c]
583
584   /-- Doubling a square's factors multiplies it by four. -/
585   theorem sq2 (a : Nat) : a * 2 * 2 = a * 4 := mulAssoc a 2 2
586
587   /-- Two copies of a doubled number make four copies. -/
588   theorem distrib4 (a : Nat) : a * 2 + a * 2 = a * 4 := (Nat.mul_add a 2 2).symm
589
590   /-- Raising the width by one site-pair multiplies the state count by four.
591   `Nat.pow_add` would give this in one line, but it is proved through
592   `propext`; the `show` below only rewrites 2(n+1) into the syntactic
593   successor form that `Nat.pow_succ` matches, which is a definitional
594   change and costs no axiom. -/
595   theorem powTwoStep (n : Nat) : 2 ^ (2 * (n + 1)) = 2 ^ (2 * n) * 4 := by
596     show 2 ^ (2 * n + 1 + 1) = 2 ^ (2 * n) * 4
597     rw [Nat.pow_succ, Nat.pow_succ, sq2]
598
599   /-- Lemma 3.1, the Burnside arithmetic, FOR EVERY N rather than at
600   tabulated widths. The fixed-point vector (2^{2N}, 0, 2^N, 2^N) verified
601   above averages over the four frame elements to exactly
602   2^{2N-2} + 2^{N-1}. Written with N = n+1 so that the exponents 2N-2 and
603   N-1 are literal and no truncated subtraction occurs. -/
604   theorem burnside_average : forall n : Nat,
605     2 ^ (2 * (n + 1)) + 0 + 2 ^ (n + 1) + 2 ^ (n + 1) = 4 * (2 ^ (2 * n) + 2 ^ n) := by
606     intro n
607     rw [powTwoStep, Nat.pow_succ, Nat.add_zero, Nat.add_assoc, distrib4 (2 ^ n),
608         Nat.mul_add, Nat.mul_comm 4 (2 ^ (2 * n)), Nat.mul_comm 4 (2 ^ n)]
609
610   /-- Corollary 3.7, the orbit decomposition, FOR EVERY N. If `f` is the
611   number of free orbits, so that f + 2^N is the total orbit count, then
612   the 2^N orbits of size two and the f orbits of size four account for
613   every one of the 2^{2N} marker assignments and for no more. The count
614   enters as a hypothesis so that no truncated subtraction appears. -/
615   theorem orbit_decomposition : forall n f : Nat,
616     f + 2 ^ n = 2 ^ (2 * n) ->
617     2 * 2 ^ (n + 1) + 4 * f = 2 ^ (2 * (n + 1)) := by
618     intro n f h
619     rw [powTwoStep, Nat.pow_succ, Nat.mul_comm 2 (2 ^ n * 2), sq2, <- h,
620         Nat.mul_comm (f + 2 ^ n) 4, Nat.mul_add 4 f (2 ^ n), Nat.mul_comm 4 (2 ^ n),
621         Nat.add_comm (2 ^ n * 4) (4 * f)]
622
623   -- =====
624   -- 5. Proposition 3.15: Trap 4, the two coincidences of counting
625   -- =====
626
627   def isSortedL : List Nat -> Bool
628   | [] => true
629   | [_] => true
630   | a :: b :: t => Nat.ble a b && isSortedL (b :: t)
631
632   /-- Proposition 3.15(i), the multiset coincidence, refuted. At N = 3 the
633   twenty classes and the twenty multisets of three letters agree in
634   number. The agreement does not persist: at N = 4 there are seventy-two
635   classes against thirty-five multisets. What holds at N = 3 is a
636   coincidence of two counting functions at one point, and carries no
637   structure. The multiset counts are obtained by enumeration, not quoted
638   as binomial coefficients. -/

```

```

639 theorem multiset_coincidence_is_isolated :
640   countP isSortedL Wd3 = 20 /\
641   C20.length = 20 /\
642   countP isSortedL Wd4 = 35 /\
643   (classesAt 8).length = 72 /\
644   !(countP isSortedL Wd4 == (classesAt 8).length) = true := by
645   repeat (first | rfl | constructor)
646
647 /-- The sorted fibre profile of the twenty classes: eight of size two and
648      twelve of size four. -/
649 def classProfile : List Nat :=
650   sortNat (C20.map (fun c => countP (fun m => clsM m == c) B6))
651
652 /-- The fibre profile of the standard genetic code, as cited: two amino
653      acids with one codon, nine with two, one with three, five with four and
654      three with six, over sixty-one sense codons with three stops outside.
655      This is an external datum, not a theorem of this file. -/
656 def codonProfile : List Nat :=
657   [1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 4, 4, 4, 4, 4, 6, 6, 6]
658
659 /-- Proposition 3.15(ii), the codon coincidence, in its CORRECTED form.
660      Counting alone does not refute the identification: the twenty classes
661      carry  $8 \times 2 + 12 \times 4 = 64$  markers, so a surjection with the right total is
662      arithmetically possible, and a draft that claimed otherwise was
663      overstated. What does refute it is the DEGENERACY PROFILE. Both
664      profiles have twenty entries; the class profile is eight 2s and twelve
665      4s summing to 64, the code profile sums to 61 with 3 stops outside, and
666      the two sorted profiles are unequal. No bijection of classes with amino
667      acids can match fibre sizes. -/
668 theorem codon_profile_refutation :
669   8 * 2 + 12 * 4 = 64 /\
670   classProfile.length = 20 /\
671   codonProfile.length = 20 /\
672   classProfile.foldl (fun a b => a + b) 0 = 64 /\
673   codonProfile.foldl (fun a b => a + b) 0 = 61 /\
674   61 + 3 = 64 /\
675   !(classProfile == codonProfile) = true := by
676   repeat (first | rfl | constructor)
677
678 -- =====
679 -- 6. Section 4: descent of a letter transformation
680 -- =====
681
682 /-- The registry partition: class 0 is {FF, BB}, class 1 is {FB, BF}. -/
683 def clsL (x : Nat) : Nat := if x == 0 || x == 3 then 0 else 1
684
685 /-- The registry classes: sequences of odd length modulo reversal. -/
686 def Xof (seqs : List (List Nat)) : List (List Nat) := nub (seqs.map Dbar)
687
688 /-- The twenty-four bijections of the four-letter alphabet, as pinned above. -/
689 abbrev P24 : List (List Nat) := P24L
690
691 /-- Letterwise application of a transformation to a word. -/
692 def apT (T : List Nat) (w : List Nat) : List Nat := w.map (fun x => T.getD x 0)
693
694 /-- Condition (i): the transformation carries the partition to itself,
695      either preserving or exchanging the two classes. -/
696 def preservesPartition (T : List Nat) : Bool :=

```

```

697 ((List.range 4).all (fun x => clsL (T.getD x 0) == clsL x)) ||
698 ((List.range 4).all (fun x => !(clsL (T.getD x 0) == clsL x)))
699
700 def preservesEachClass (T : List Nat) : Bool :=
701   (List.range 4).all (fun x => clsL (T.getD x 0) == clsL x)
702
703 /-- The eight admissible transformations, as pinned above. -/
704 abbrev ADM : List (List Nat) := ADML
705
706 /-- Conjugation of a letter map by a transformation:  $T \mapsto T^{-1}$ . -/
707 def conjT (T : List Nat) (g : List Nat) : List Nat :=
708   (List.range 4).map (fun y =>
709     T.getD (g.getD ((List.range 4).findIdx (fun k => T.getD k 0 == y)) 0) 0)
710
711 def rhoMap : List Nat := (List.range 4).map rhoL
712 def tauMap : List Nat := (List.range 4).map tauL
713 def idMap : List Nat := (List.range 4).map (fun x => x)
714 def tauRho : List Nat := (List.range 4).map (fun x => rhoL (tauL x))
715
716 /-- The least element of a list of words under the numeral order, used as
717   the canonical name of a frame class of words. -/
718 def minTo (best : List Nat) : List (List Nat) -> List Nat
719   | [] => best
720   | a :: t => minTo (if Nat.ble (toNat a) (toNat best) then a else best) t
721
722 def minOf : List (List Nat) -> List Nat
723   | [] => []
724   | a :: t => minTo a t
725
726 def orbitW (w : List Nat) : List (List Nat) :=
727   nub [w, flipW w, revW w, flipW (revW w)]
728 def classOfW (w : List Nat) : List Nat := minOf (orbitW w)
729 def classesW (ws : List (List Nat)) : List (List Nat) := nub (ws.map classOfW)
730
731 abbrev CW1 : List (List Nat) := CW1L
732 abbrev CW2 : List (List Nat) := CW2L
733 abbrev CW3 : List (List Nat) := CW3L
734
735 /-- The word classification agrees with the marker classification of
736   Section 3, as Lemma 2.15 requires: 2, 6, 20 and 72 classes of words of
737   length 1, 2, 3 and 4. This is the check that detects a canonical
738   representative chosen in the wrong base, which would split the twenty
739   classes into twenty-two without disturbing any other count here. -/
740 theorem word_classes_match_marker_classes :
741   CW1.length = 2 /\ CW2.length = 6 /\ CW3.length = 20 /\
742   CW3.length = C20.length := by
743   repeat (first | rfl | constructor)
744
745 /-- The pinned enumerations are exactly the derived ones.
746
747   This is checked by the CHARACTERISTIC PROPERTY of a classification
748   rather than by recomputing it and comparing. For a list L to be the
749   classification of a source S under a canonicalising map f, three things
750   suffice: every entry of L is canonical (f fixes it), every element of S
751   has its image in L (completeness), and L has no repetitions. Each of
752   those is linear or quadratic in data that is already given, whereas
753   recomputing `nub (S.map f)` and comparing re-derives the classification
754   inside the comparison -- which does not finish here.

```

```

755
756   The property form is also stronger in one respect and weaker in none:
757   it does not depend on the order in which the enumeration happens to
758   produce elements, so it cannot be satisfied by accident of ordering and
759   cannot fail for want of it. -/
760 theorem pinned_lists_are_the_derived_ones :
761   -- the twenty registry classes at 2N = 6
762   (C20L.all (fun c => Dbar c == c)) = true /\
763   ((allBits 6).all (fun m => C20L.contains (clsM m))) = true /\
764   (nub C20L == C20L) = true /\ C20L.length = 20 /\
765   -- the word classifications at three widths
766   (CW1L.all (fun c => classOfW c == c)) = true /\
767   (Wd1.all (fun w => CW1L.contains (classOfW w))) = true /\
768   (nub CW1L == CW1L) = true /\
769   (CW2L.all (fun c => classOfW c == c)) = true /\
770   (Wd2.all (fun w => CW2L.contains (classOfW w))) = true /\
771   (nub CW2L == CW2L) = true /\
772   (CW3L.all (fun c => classOfW c == c)) = true /\
773   (Wd3.all (fun w => CW3L.contains (classOfW w))) = true /\
774   (nub CW3L == CW3L) = true /\
775   -- the twenty-four bijections and the eight admissible ones
776   (P24L.all (fun t =>
777     t.contains 0 && t.contains 1 && t.contains 2 && t.contains 3)) = true /\
778   (Wd4.all (fun t =>
779     if t.contains 0 && t.contains 1 && t.contains 2 && t.contains 3
780     then P24L.contains t else true)) = true /\
781   (nub P24L == P24L) = true /\ P24L.length = 24 /\
782   (P24L.filter preservesPartition == ADML) = true /\
783   -- the registry classes at N = 2, 3, 4
784   (X2L.all (fun c => Dbar c == c)) = true /\
785   (S3.all (fun d => X2L.contains (Dbar d))) = true /\
786   (nub X2L == X2L) = true /\
787   (X3L.all (fun c => Dbar c == c)) = true /\
788   (S5.all (fun d => X3L.contains (Dbar d))) = true /\
789   (nub X3L == X3L) = true /\
790   (X4L.all (fun c => Dbar c == c)) = true /\
791   (S7.all (fun d => X4L.contains (Dbar d))) = true /\
792   (nub X4L == X4L) = true /\ X4L.length = 72 := by
793   repeat (first | rfl | constructor)
794
795 /-- Lemma 4.5. Exactly eight of the twenty-four bijections are isomorphisms
796   of structured alphabets; four preserve each class of the partition and
797   four exchange the two classes. -/
798 theorem eight_isomorphisms :
799   P24.length = 24 /\
800   ADM.length = 8 /\
801   (ADM.filter preservesEachClass).length = 4 /\
802   (ADM.filter (fun T => !preservesEachClass T)).length = 4 := by
803   repeat (first | rfl | constructor)
804
805 /-- Condition (ii): conjugation carries the structural group to itself. -/
806 def condII (T : List Nat) : Bool :=
807   [idMap, rhoMap, tauMap, tauRho].all (fun g => preservesEachClass (conjT T g))
808
809 /-- Condition (iii): conjugation fixes complementation. -/
810 def condIII (T : List Nat) : Bool := conjT T rhoMap == rhoMap
811
812 /-- Condition (iv): INTERTWINING of the two frame-group actions. The

```

```

813 transformation commutes with flip, and with reversal up to flip. This
814 is the correct form; the literal generator-matching form (iv') below is
815 strictly stronger and is NOT equivalent to the others. -/
816 def condIV (T : List Nat) (ws : List (List Nat)) : Bool :=
817   (ws.all (fun w => apT T (flipW w) == flipW (apT T w))) &&
818   ((ws.all (fun w => apT T (revW w) == revW (apT T w))) ||
819    (ws.all (fun w => apT T (revW w) == flipW (revW (apT T w)))))
820
821 /-- Condition (iv'): literal equivariance for the fixed labelling. -/
822 def condIVp (T : List Nat) (ws : List (List Nat)) : Bool :=
823   (ws.all (fun w => apT T (flipW w) == flipW (apT T w))) &&
824   (ws.all (fun w => apT T (revW w) == revW (apT T w)))
825
826 /-- Condition (v): descent to the orbit sets. The image of each orbit
827 must BE an orbit. Asking only that the image of a representative land
828 in some class would be vacuous -- every word lies in some class -- so
829 the image set is compared with the orbit of the image representative,
830 both ways. -/
831 def condV (T : List Nat) (cs : List (List Nat)) : Bool :=
832   cs.all (fun c =>
833     ((orbitW (apT T c)).length == (orbitW c).length) &&
834     ((orbitW c).all (fun w => (orbitW (apT T c)).contains (apT T w))))
835
836 /-- The five conditions agree at every one of the twenty-four bijections,
837 for the given width. -/
838 def tfaeOK (ws cs : List (List Nat)) : Bool :=
839   P24.all (fun T =>
840     (preservesPartition T == condII T) &&
841     (preservesPartition T == condIII T) &&
842     (preservesPartition T == condIV T ws) &&
843     (preservesPartition T == condV T cs))
844
845 /-- Theorem 4.9. For every one of the twenty-four bijections and for
846 N = 1, 2, 3, the five conditions (i)--(v) take the same truth value.
847 The equivalence is checked exhaustively, not at a sample: every
848 transformation, at three widths, every condition evaluated on every
849 word. -/
850 theorem descent_conditions_equivalent :
851   (tfaeOK Wd1 CW1 && tfaeOK Wd2 CW2 && tfaeOK Wd3 CW3) = true := by rfl
852
853 /-- Proposition 4.14. Exactly eight of the twenty-four bijections descend. -/
854 theorem exactly_eight_descend :
855   (P24.filter (fun T => condV T CW3)).length = 8 /\
856   (P24.filter (fun T => condV T CW2)).length = 8 /\
857   ADM.length = 8 := by
858   repeat (first | rfl | constructor)
859
860 /-- (iv') holds exactly for the admissible transformations preserving each
861 class, at the given width. -/
862 def ivpOK (ws cs : List (List Nat)) : Bool :=
863   P24.all (fun T => condIVp T ws == (condV T cs && preservesEachClass T))
864
865 /-- Proposition 4.11. The literal condition (iv') is STRICTLY STRONGER than
866 the other four. For N >= 2 it holds for exactly the four admissible
867 transformations that preserve each class, and fails for the four that
868 exchange them: four of twenty-four against eight. The theorem as first
869 drafted, with (iv') in place of (iv), would have been false. -/
870 theorem ivprime_strictly_stronger :

```

```

871 (ivpOK Wd2 CW2 && ivpOK Wd3 CW3) = true /\
872 (P24.filter (fun T => condIVp T Wd3)).length = 4 /\
873 (P24.filter (fun T => condV T CW3)).length = 8 /\
874 !((P24.filter (fun T => condIVp T Wd3)).length ==
875   (P24.filter (fun T => condV T CW3)).length) = true := by
876 repeat (first | rfl | constructor)
877
878 /-- The witness of Proposition 4.11: the class-exchanging transformation
879 (FF FB)(BB BF). It is admissible and commutes with complementation, yet
880 it conjugates tau to tau rho rather than to tau -- which is exactly why
881 intertwining holds for it and literal generator-matching does not. -/
882 theorem ivprime_witness :
883   preservesPartition [1, 0, 3, 2] = true /\
884   preservesEachClass [1, 0, 3, 2] = false /\
885   conjT [1, 0, 3, 2] rhoMap = rhoMap /\
886   conjT [1, 0, 3, 2] tauMap = tauRho /\
887   condIV [1, 0, 3, 2] Wd3 = true /\
888   condIVp [1, 0, 3, 2] Wd3 = false := by
889 repeat (first | rfl | constructor)
890
891 /-- Theorem 4.9, the size clause. Each of the eight admissible
892 transformations induces a bijection of the classes preserving class
893 size. -/
894 theorem descent_preserves_class_size :
895   (ADM.all (fun T => CW3.all (fun c => (orbitW (apT T c)).length == (orbitW c).length)))
896   = true := by rfl
897
898 /-- Example 4.15, the counterexample, as the article states it. The
899 transformation (BB BF), which does not preserve the partition, carries
900 ONE class of size four onto a set of four words lying in FOUR DISTINCT
901 classes, of sizes 2, 2, 4, 4. A transformation that fails the condition
902 does not merely distort the classification; it destroys it. -/
903 theorem counterexample_meets_four_classes :
904   preservesPartition [0, 1, 3, 2] = false /\
905   (orbitW [0, 1, 2]).length = 4 /\
906   (nub ((orbitW [0, 1, 2]).map (fun w => classOfW (apT [0, 1, 3, 2] w))))).length = 4 /\
907   sortNat ((orbitW [0, 1, 2]).map (fun w => (orbitW (apT [0, 1, 3, 2] w)).length))
908   = [2, 2, 4, 4] := by
909 repeat (first | rfl | constructor)
910
911 -- =====
912 -- 7. Section 5: the induced action on the classes
913 -- =====
914
915 /-- How many admissible transformations act trivially on the given classes. -/
916 def kerCount (cs : List (List Nat)) : Nat :=
917   (ADM.filter (fun T => cs.all (fun c => classOfW (apT T c) == c))).length
918
919 /-- Proposition 5.3(i). The kernel of the action of the admissible
920 transformations on the classes is {id, rho} for N = 2, 3, and is the
921 whole structural group at N = 1. The identity and complementation act
922 trivially; nothing else does. -/
923 theorem kernel_is_id_and_rho :
924   kerCount CW2 = 2 /\
925   kerCount CW3 = 2 /\
926   kerCount CW1 = 4 /\
927   (CW3.all (fun c => classOfW (apT rhoMap c) == c)) = true /\
928   (CW3.all (fun c => classOfW (apT idMap c) == c)) = true := by

```

```

929   repeat (first | rfl | constructor)
930
931   /-- The induced map of an admissible transformation on the twenty classes,
932       recorded as the list of numerals of the image classes. -/
933   def inducedOn (T : List Nat) : List Nat := CW3.map (fun c => toNat (classOfW (apT T c)))
934
935   /-- The distinct induced maps. -/
936   def INDUCED : List (List Nat) := nub (ADM.map inducedOn)
937
938   /-- Proposition 5.3(iii). The eight admissible transformations induce only
939       FOUR distinct maps of the twenty classes, each fibre of size two. No
940       orbit-level quantity can therefore separate more than four of them,
941       which is the obstruction the whole of Section 5 is about. -/
942   theorem psi_is_two_to_one :
943     INDUCED.length = 4 /\
944     ADM.length = 8 /\
945     (INDUCED.all (fun img => (ADM.filter (fun T => inducedOn T == img)).length == 2))
946     = true := by
947     repeat (first | rfl | constructor)
948
949   /-- Proposition 5.5. kappa leaves the odd-indexed entries of the registry
950       sequence alone and replaces each even-indexed entry by the sum of it and
951       its two neighbours. -/
952   def kappaD (d : List Nat) : List Nat :=
953     (List.range d.length).map (fun i =>
954       if i % 2 == 1 then (d.getD (i - 1) 0 + d.getD i 0 + d.getD (i + 1) 0) % 2
955       else d.getD i 0)
956
957   /-- lambda is complementation of the registry sequence. -/
958   def lambdaD (d : List Nat) : List Nat := d.map (fun x => (x + 1) % 2)
959
960   abbrev X2 : List (List Nat) := X2L
961   abbrev X3 : List (List Nat) := X3L
962   abbrev X4 : List (List Nat) := X4L
963
964   /-- kappa and lambda descend to the classes and carry classes to classes. -/
965   def wdOK (seqs cls : List (List Nat)) : Bool :=
966     (seqs.all (fun d =>
967       (Dbar (kappaD d.reverse) == Dbar (kappaD d)) &&
968       (Dbar (lambdaD d.reverse) == Dbar (lambdaD d)))) &&
969     (cls.all (fun c => cls.contains (Dbar (kappaD c)) && cls.contains (Dbar (lambdaD c))))
970
971   /-- Proposition 5.5, well-definedness. kappa and lambda descend to the
972       classes: each commutes with reversal up to reversal, so the image class
973       does not depend on the representative chosen, and each carries classes
974       to classes. Checked at N = 2, 3, 4.
975
976       N = 5 is deliberately not included anywhere in this file, and the
977       omission is recorded rather than hidden. The 272 classes at N = 5 are
978       built from 512 sequences, and the quantifiers over them exceed the
979       memory of the machine this dossier was built on; what is lost is a
980       second odd-N data point, not a claim. The same applies to
981       `lambda_acts_freely` and `kappa_central_entry`. -/
982   theorem kappa_lambda_well_defined :
983     (wdOK S3 X2 && wdOK S5 X3 && wdOK S7 X4) = true := by rfl
984
985   /-- Proposition 5.5, the freeness of lambda -- VERIFIED, NOT ASSERTED. The
986       class-inverting element acts with no fixed class at N = 2, 3, 4. The

```

```

987 contrast with the other two elements is the point: kappa has 2^N fixed
988 classes at every N, and kappa lambda has fixed classes exactly when N is
989 even -- two at N = 2 and eight at N = 4, none at N = 3. An
990 argument that treated the three non-identity elements alike would be
991 wrong, and an earlier draft of the article did. -/
992 theorem lambda_acts_freely :
993   countP (fun c => Dbar (lambdaD c) == c) X2 = 0 /\
994   countP (fun c => Dbar (lambdaD c) == c) X3 = 0 /\
995   countP (fun c => Dbar (lambdaD c) == c) X4 = 0 /\
996   countP (fun c => Dbar (kappaD c) == c) X2 = 4 /\
997   countP (fun c => Dbar (kappaD c) == c) X3 = 8 /\
998   countP (fun c => Dbar (kappaD c) == c) X4 = 16 /\
999   countP (fun c => Dbar (lambdaD (kappaD c)) == c) X2 = 2 /\
1000   countP (fun c => Dbar (lambdaD (kappaD c)) == c) X3 = 0 /\
1001   countP (fun c => Dbar (lambdaD (kappaD c)) == c) X4 = 8 := by
1002   repeat (first | rfl | constructor)
1003
1004 /-- kappa preserves the central entry on the reversal-symmetric classes. -/
1005 def kapSymOK (n : Nat) (cls : List (List Nat)) : Bool :=
1006   cls.all (fun c =>
1007     if c == c.reverse then (kappaD c).getD (n - 1) 0 == c.getD (n - 1) 0 else true)
1008
1009 /-- Proposition 5.5, the central entry. kappa preserves the central entry
1010 of the registry sequence on every reversal-symmetric class, at every N
1011 checked; but on EVERY class exactly when N is odd. The last two clauses
1012 are the counterexamples at N = 2 and N = 4, so the even case is a
1013 genuine failure and not an artefact of the representative. -/
1014 theorem kappa_central_entry :
1015   (kapSymOK 2 X2 && kapSymOK 3 X3 && kapSymOK 4 X4) = true /\
1016   (X3.all (fun c => (kappaD c).getD 2 0 == c.getD 2 0)) = true /\
1017   (X2.all (fun c => (kappaD c).getD 1 0 == c.getD 1 0)) = false /\
1018   (X4.all (fun c => (kappaD c).getD 3 0 == c.getD 3 0)) = false := by
1019   repeat (first | rfl | constructor)
1020
1021 /-- Proposition 5.5(i). The kappa formula is not a definition made to fit:
1022 it IS the letterwise action of tau, transported through W. For each of
1023 the sixty-four marker assignments there is exactly one assignment whose
1024 word is tau applied letterwise, and its registry sequence is exactly
1025 kappa of the original. -/
1026 theorem kappa_is_the_tau_action :
1027   (B6.all (fun m => (B6.filter (fun mm => W mm == (W m).map tauL)).length == 1))
1028   = true /\
1029   (B6.all (fun m =>
1030     (B6.filter (fun mm => W mm == (W m).map tauL)).all (fun mm =>
1031       Dseq mm == kappaD (Dseq m))))
1032   = true := by
1033   repeat (first | rfl | constructor)
1034
1035 /-- Theorem 5.8(i). Every admissible transformation commutes with
1036 complementation, the conjugate of rho being rho itself. The residual
1037 indeterminacy of the section is therefore exactly the coset of rho: a
1038 two-element ambiguity and not a larger one. -/
1039 theorem admissible_commutes_with_rho :
1040   (ADM.all (fun T =>
1041     (conjT T rhoMap == rhoMap) &&
1042     ((List.range 4).all (fun x => T.getD (rhoL x) 0 == rhoL (T.getD x 0)))))
1043   = true := by rfl
1044

```

```

1045 -- =====
1046 -- 8. Section 6: the variational criterion
1047 --
1048 -- The functional of Definition 6.6 takes dyadic values in the worked
1049 -- examples: the ensemble is uniform on two classes, so the denominator
1050 -- is 2 throughout. Clearing it turns every comparison into an identity
1051 -- of natural numbers, which is what is proved below. Nothing is lost:
1052 -- a maximiser of Phi is a maximiser of 2 Phi.
1053 -- =====
1054
1055 /-- The four induced maps on the classes: identity, kappa, lambda, kappa
1056 lambda. -/
1057 def indMap (k : Nat) (c : List Nat) : List Nat :=
1058   if k == 0 then c
1059   else if k == 1 then Dbar (kappaD c)
1060   else if k == 2 then Dbar (lambdaD c)
1061   else Dbar (lambdaD (kappaD c))
1062
1063 /-- A weight, given as a list of class-value pairs and read as a function
1064 vanishing off that list. -/
1065 def wAt (w : List (Prod (List Nat) Nat)) (c : List Nat) : Nat :=
1066   (w.filter (fun p => p.1 == c)).foldl (fun a p => a + p.2) 0
1067
1068 /-- Definition 6.6, cleared of denominators: the ensemble is a list of
1069 classes each of unit weight. -/
1070 def Phi (mu : List (List Nat)) (w : List (Prod (List Nat) Nat)) (k : Nat) : Nat :=
1071   mu.foldl (fun a c => a + wAt w (indMap k c)) 0
1072
1073 def mu0 : List (List Nat) := [[0, 0, 0, 0, 0], [0, 0, 1, 0, 1]]
1074
1075 def w614 : List (Prod (List Nat) Nat) :=
1076   [[0, 0, 1, 0, 1], 1], ([0, 1, 1, 0, 1], 2), ([0, 1, 0, 1, 1], 4), ([0, 1, 0, 0, 1], 8)]
1077 def w615 : List (Prod (List Nat) Nat) :=
1078   [[0, 0, 1, 0, 1], 1], ([0, 1, 0, 1, 1], 1)]
1079 def w616 : List (Prod (List Nat) Nat) :=
1080   [[0, 0, 1, 0, 1], 1], ([0, 1, 1, 0, 1], 1), ([0, 1, 0, 1, 1], 5), ([0, 0, 0, 0, 1], 3)]
1081
1082 /-- How many of the four values attain the stated maximum. -/
1083 def argmaxCount (w : List (Prod (List Nat) Nat)) (best : Nat) : Nat :=
1084   ((List.range 4).filter (fun k => Phi mu0 w k == best)).length
1085
1086 /-- Example 6.14. A weight for which the four values of the functional are
1087 1, 2, 4, 8 (that is 1/2, 1, 2, 4 before clearing), so the maximiser is
1088 unique and is kappa lambda. Off the degeneracy locus the criterion
1089 selects a single transformation up to the residual coset. -/
1090 theorem phi_unique_maximiser :
1091   Phi mu0 w614 0 = 1 /\ Phi mu0 w614 1 = 2 /\
1092   Phi mu0 w614 2 = 4 /\ Phi mu0 w614 3 = 8 /\
1093   argmaxCount w614 8 = 1 := by
1094   repeat (first | rfl | constructor)
1095
1096 /-- Example 6.15. A lambda-invariant weight ties the four values in two
1097 pairs: the functional cannot then distinguish a transformation from its
1098 composite with the class-inverting element. This is a point of the
1099 degeneracy locus arising from a symmetry of the weight. -/
1100 theorem phi_tie_from_symmetry :
1101   (C20.all (fun c => wAt w615 (Dbar (lambdaD c)) == wAt w615 c)) = true /\
1102   Phi mu0 w615 0 = 1 /\ Phi mu0 w615 1 = 0 /\

```

```

1103   Phi mu0 w615 2 = 1 /\ Phi mu0 w615 3 = 0 /\
1104   Phi mu0 w615 0 = Phi mu0 w615 2 /\ Phi mu0 w615 1 = Phi mu0 w615 3 := by
1105   repeat (first | rfl | constructor)
1106
1107   /-- Remark 6.17, the witness that the degeneracy locus is STRICTLY LARGER
1108   than the locus of non-unique maximisers. Here two of the four values
1109   coincide, so the point lies on Z; the maximiser is nevertheless unique;
1110   and neither the ensemble nor the weight carries a symmetry. Degeneracy
1111   of the functional is therefore not explained by symmetry of its data,
1112   which is the content of the remark. -/
1113   theorem Z_strictly_larger_than_Zmax :
1114     Phi mu0 w616 0 = 1 /\ Phi mu0 w616 1 = 1 /\
1115     Phi mu0 w616 2 = 5 /\ Phi mu0 w616 3 = 0 /\
1116     Phi mu0 w616 0 = Phi mu0 w616 1 /\
1117     argmaxCount w616 5 = 1 /\
1118     (C20.all (fun c => wAt w616 (Dbar (lambdaD c)) == wAt w616 c)) = false /\
1119     (C20.all (fun c => wAt w616 (Dbar (kappaD c)) == wAt w616 c)) = false := by
1120     repeat (first | rfl | constructor)
1121
1122   /-- Proposition 6.11. The four induced maps are pairwise distinct as maps
1123   of the twenty classes. Each of the six pair conditions defining the
1124   degeneracy locus is therefore a nonzero bilinear form, and Z is a proper
1125   closed subset rather than the whole parameter space. -/
1126   theorem four_induced_maps_distinct :
1127     (nub ((List.range 4).map (fun k => C20.map (fun c => toNat (indMap k c))))).length = 4 := by
1128     rfl
1129
1130   /-- Remark 6.18. The class-inverting element has ten orbits, all of size
1131   two, on the twenty classes -- which is the dimension the remark records
1132   for the space of weights tying the functional in pairs. -/
1133   theorem lambda_has_ten_orbits :
1134     (nub (C20.map (fun c => minOf (nub [c, Dbar (lambdaD c)])))).length = 10 /\
1135     (C20.all (fun c => (nub [c, Dbar (lambdaD c)]).length == 2)) = true := by
1136     repeat (first | rfl | constructor)
1137
1138   -- =====
1139   -- 9. Appendix A: the printed table
1140   -- =====
1141
1142   /-- One row of Table A.1: representative, class label, class size,
1143   stabiliser type, central entry, and the images under kappa and lambda. -/
1144   structure Row where
1145     rep : List Nat
1146     lbl : List Nat
1147     size : Nat
1148     stab : Nat
1149     cen : Nat
1150     kap : List Nat
1151     lam : List Nat
1152   deriving BEq
1153
1154   /-- The twenty rows exactly as printed in Appendix A, with F written 0 and
1155   B written 1 in the representative, and the stabiliser types 1 trivial,
1156   2 palindromic, 3 anti-palindromic. -/
1157   def tableA : List Row :=
1158   [
1159     Row.mk [0, 0, 0, 0, 0, 0] [0, 0, 0, 0, 0] 2 2 0 [0, 0, 0, 0, 0] [1, 1, 1, 1, 1],
1160     Row.mk [0, 0, 0, 0, 0, 1] [0, 0, 0, 0, 1] 4 1 0 [0, 0, 0, 1, 1] [0, 1, 1, 1, 1],

```

```

1161 Row.mk [0, 0, 0, 0, 1, 1] [0, 0, 0, 1, 0] 4 1 0 [0, 0, 0, 1, 0] [1, 0, 1, 1, 1],
1162 Row.mk [0, 0, 0, 0, 1, 0] [0, 0, 0, 1, 1] 4 1 0 [0, 0, 0, 0, 1] [0, 0, 1, 1, 1],
1163 Row.mk [0, 0, 0, 1, 1, 1] [0, 0, 1, 0, 0] 2 3 1 [0, 1, 1, 1, 0] [1, 1, 0, 1, 1],
1164 Row.mk [0, 0, 0, 1, 1, 0] [0, 0, 1, 0, 1] 4 1 1 [0, 1, 1, 0, 1] [0, 1, 0, 1, 1],
1165 Row.mk [0, 0, 0, 1, 0, 0] [0, 0, 1, 1, 0] 4 1 1 [0, 0, 1, 1, 0] [1, 0, 0, 1, 1],
1166 Row.mk [0, 0, 0, 1, 0, 1] [0, 0, 1, 1, 1] 4 1 1 [0, 1, 1, 1, 1] [0, 0, 0, 1, 1],
1167 Row.mk [0, 0, 1, 1, 1, 0] [0, 1, 0, 0, 1] 4 1 0 [0, 1, 0, 1, 1] [0, 1, 1, 0, 1],
1168 Row.mk [0, 0, 1, 1, 0, 0] [0, 1, 0, 1, 0] 2 2 0 [0, 1, 0, 1, 0] [1, 0, 1, 0, 1],
1169 Row.mk [0, 0, 1, 1, 0, 1] [0, 1, 0, 1, 1] 4 1 0 [0, 1, 0, 0, 1] [0, 0, 1, 0, 1],
1170 Row.mk [0, 0, 1, 0, 0, 1] [0, 1, 1, 0, 1] 4 1 1 [0, 0, 1, 0, 1] [0, 1, 0, 0, 1],
1171 Row.mk [0, 0, 1, 0, 1, 1] [0, 1, 1, 1, 0] 2 3 1 [0, 0, 1, 0, 0] [1, 0, 0, 0, 1],
1172 Row.mk [0, 0, 1, 0, 1, 0] [0, 1, 1, 1, 1] 4 1 1 [0, 0, 1, 1, 1] [0, 0, 0, 0, 1],
1173 Row.mk [0, 1, 1, 1, 1, 0] [1, 0, 0, 0, 1] 2 2 0 [1, 1, 0, 1, 1] [0, 1, 1, 1, 0],
1174 Row.mk [0, 1, 0, 0, 0, 1] [1, 0, 0, 1, 1] 4 1 0 [1, 0, 0, 1, 1] [0, 0, 1, 1, 0],
1175 Row.mk [0, 1, 1, 0, 0, 1] [1, 0, 1, 0, 1] 2 3 1 [1, 0, 1, 0, 1] [0, 1, 0, 1, 0],
1176 Row.mk [0, 1, 0, 1, 1, 0] [1, 0, 1, 1, 1] 4 1 1 [1, 0, 1, 1, 1] [0, 0, 0, 1, 0],
1177 Row.mk [0, 1, 0, 0, 1, 0] [1, 1, 0, 1, 1] 2 2 0 [1, 0, 0, 0, 1] [0, 0, 1, 0, 0],
1178 Row.mk [0, 1, 0, 1, 0, 1] [1, 1, 1, 1, 1] 2 3 1 [1, 1, 1, 1, 1] [0, 0, 0, 0, 0]
1179 ]
1180
1181 /-- The row this file computes for a given representative. -/
1182 def rowOf (m : List Nat) : Row :=
1183   Row.mk m (clsM m) (orbitM m).length (stabType m) ((clsM m).getD 2 0)
1184   (Dbar (kappaD (clsM m))) (Dbar (lambdaD (clsM m)))
1185
1186 /-- Appendix A. Every one of the twenty printed rows is recomputed from the
1187   definitions and agrees field by field. The table is also COMPLETE and
1188   without repetition: the twenty printed labels are distinct, and every
1189   one of the twenty classes appears among them, so nothing was printed
1190   twice and nothing was left out. -/
1191 theorem tableA_agrees :
1192   tableA.length = 20 /\
1193   (tableA.all (fun r => rowOf r.rep == r)) = true /\
1194   (nub (tableA.map (fun r => r.lbl))).length = 20 /\
1195   (C20.all (fun c => (tableA.map (fun r => r.lbl)).contains c)) = true := by
1196   repeat (first | rfl | constructor)
1197
1198 /-- Appendix A, the asterisks. Exactly eight of the twenty classes are
1199   fixed by kappa, which is the 2^N of Proposition 5.5 at N = 3. -/
1200 theorem tableA_kappa_fixed :
1201   (tableA.filter (fun r => r.kap == r.lbl)).length = 8 /\
1202   (8 : Nat) = 2 ^ 3 := by
1203   repeat (first | rfl | constructor)
1204
1205 end ChryseneFormalism.Article45
1206
1207 -- Kernel Axiom Audit:
1208 #print axioms ChryseneFormalism.Article45.frame_actions_agree
1209 #print axioms ChryseneFormalism.Article45.witness_FFFBBF
1210 #print axioms ChryseneFormalism.Article45.orbit_count_enumerated
1211 #print axioms ChryseneFormalism.Article45.fixed_point_vector
1212 #print axioms ChryseneFormalism.Article45.flip_is_fixed_point_free
1213 #print axioms ChryseneFormalism.Article45.Dbar_separates_orbits
1214 #print axioms ChryseneFormalism.Article45.orbit_sizes_two_or_four
1215 #print axioms ChryseneFormalism.Article45.stabiliser_central_entry
1216 #print axioms ChryseneFormalism.Article45.type_counts_at_six
1217 #print axioms ChryseneFormalism.Article45.mulAssoc
1218 #print axioms ChryseneFormalism.Article45.sq2

```

```

1219 #print axioms ChryseneFormalism.Article45.distrib4
1220 #print axioms ChryseneFormalism.Article45.powTwoStep
1221 #print axioms ChryseneFormalism.Article45.burnside_average
1222 #print axioms ChryseneFormalism.Article45.orbit_decomposition
1223 #print axioms ChryseneFormalism.Article45.multiset_coincidence_is_isolated
1224 #print axioms ChryseneFormalism.Article45.codon_profile_refutation
1225 #print axioms ChryseneFormalism.Article45.pinned_lists_are_the_derived_ones
1226 #print axioms ChryseneFormalism.Article45.eight_isomorphisms
1227 #print axioms ChryseneFormalism.Article45.word_classes_match_marker_classes
1228 #print axioms ChryseneFormalism.Article45.descent_conditions_equivalent
1229 #print axioms ChryseneFormalism.Article45.exactly_eight_descend
1230 #print axioms ChryseneFormalism.Article45.ivprime_strictly_stronger
1231 #print axioms ChryseneFormalism.Article45.ivprime_witness
1232 #print axioms ChryseneFormalism.Article45.descent_preserves_class_size
1233 #print axioms ChryseneFormalism.Article45.counterexample_meets_four_classes
1234 #print axioms ChryseneFormalism.Article45.kernel_is_id_and_rho
1235 #print axioms ChryseneFormalism.Article45.psi_is_two_to_one
1236 #print axioms ChryseneFormalism.Article45.kappa_lambda_well_defined
1237 #print axioms ChryseneFormalism.Article45.lambda_acts_freely
1238 #print axioms ChryseneFormalism.Article45.kappa_central_entry
1239 #print axioms ChryseneFormalism.Article45.kappa_is_the_tau_action
1240 #print axioms ChryseneFormalism.Article45.admissible_commutates_with_rho
1241 #print axioms ChryseneFormalism.Article45.phi_unique_maximiser
1242 #print axioms ChryseneFormalism.Article45.phi_tie_from_symmetry
1243 #print axioms ChryseneFormalism.Article45.Z_strictly_larger_than_Zmax
1244 #print axioms ChryseneFormalism.Article45.four_induced_maps_distinct
1245 #print axioms ChryseneFormalism.Article45.lambda_has_ten_orbits
1246 #print axioms ChryseneFormalism.Article45.tableA_agrees
1247 #print axioms ChryseneFormalism.Article45.tableA_kappa_fixed

```

Listing 1: Complete machine-checked Lean 4 formalization FormalProof_A45.lean for Article 45: 1247 lines, Core Lean only.

4 Lean 4 Kernel Axiom Audit Log

The following is the verbatim standard output of the compiler, reproduced without edit. The command terminated with exit status 0 and emitted no warning and no error; the 40 lines below are therefore the whole of its output.

Toolchain. \$ lean --version

Lean (version 4.14.0, x86_64-unknown-linux-gnu, commit 410fab728470, Release)

Compile and audit. \$ lean FormalProof_A45.lean

```

'ChryseneFormalism.Article45.frame_actions_agree' does not depend on any axioms
'ChryseneFormalism.Article45.witness_FFFBBF' does not depend on any axioms
'ChryseneFormalism.Article45.orbit_count_enumerated' does not depend on any axioms
'ChryseneFormalism.Article45.fixed_point_vector' does not depend on any axioms
'ChryseneFormalism.Article45.flip_is_fixed_point_free' does not depend on any axioms
'ChryseneFormalism.Article45.Dbar_separates_orbits' does not depend on any axioms
'ChryseneFormalism.Article45.orbit_sizes_two_or_four' does not depend on any axioms
'ChryseneFormalism.Article45.stabiliser_central_entry' does not depend on any axioms
'ChryseneFormalism.Article45.type_counts_at_six' does not depend on any axioms
'ChryseneFormalism.Article45.mulAssoc' does not depend on any axioms
'ChryseneFormalism.Article45.sq2' does not depend on any axioms
'ChryseneFormalism.Article45.distrib4' does not depend on any axioms
'ChryseneFormalism.Article45.powTwoStep' does not depend on any axioms

```

```

'ChryseneFormalism.Article45.burnside_average' does not depend on any axioms
'ChryseneFormalism.Article45.orbit_decomposition' does not depend on any axioms
'ChryseneFormalism.Article45.multiset_coincidence_is_isolated' does not depend on any axioms
'ChryseneFormalism.Article45.codon_profile_refutation' does not depend on any axioms
'ChryseneFormalism.Article45.pinned_lists_are_the_derived_ones' does not depend on any axioms
'ChryseneFormalism.Article45.eight_isomorphisms' does not depend on any axioms
'ChryseneFormalism.Article45.word_classes_match_marker_classes' does not depend on any axioms
'ChryseneFormalism.Article45.descent_conditions_equivalent' does not depend on any axioms
'ChryseneFormalism.Article45.exactly_eight_descend' does not depend on any axioms
'ChryseneFormalism.Article45.ivprime_strictly_stronger' does not depend on any axioms
'ChryseneFormalism.Article45.ivprime_witness' does not depend on any axioms
'ChryseneFormalism.Article45.descent_preserves_class_size' does not depend on any axioms
'ChryseneFormalism.Article45.counterexample_meets_four_classes' does not depend on any axioms
'ChryseneFormalism.Article45.kernel_is_id_and_rho' does not depend on any axioms
'ChryseneFormalism.Article45.psi_is_two_to_one' does not depend on any axioms
'ChryseneFormalism.Article45.kappa_lambda_well_defined' does not depend on any axioms
'ChryseneFormalism.Article45.lambda_acts_freely' does not depend on any axioms
'ChryseneFormalism.Article45.kappa_central_entry' does not depend on any axioms
'ChryseneFormalism.Article45.kappa_is_the_tau_action' does not depend on any axioms
'ChryseneFormalism.Article45.admissible_commutes_with_rho' does not depend on any axioms
'ChryseneFormalism.Article45.phi_unique_maximiser' does not depend on any axioms
'ChryseneFormalism.Article45.phi_tie_from_symmetry' does not depend on any axioms
'ChryseneFormalism.Article45.Z_strictly_larger_than_Zmax' does not depend on any axioms
'ChryseneFormalism.Article45.four_induced_maps_distinct' does not depend on any axioms
'ChryseneFormalism.Article45.lambda_has_ten_orbits' does not depend on any axioms
'ChryseneFormalism.Article45.tableA_agrees' does not depend on any axioms
'ChryseneFormalism.Article45.tableA_kappa_fixed' does not depend on any axioms

```

Exit code 0. Every theorem reports does not depend on any axioms: not propext, not Quot.sound, not Classical.choice, and not sorryAx.

How to read this log, and how to check it. The phrase *does not depend on any axioms* is Lean’s strongest report. Its weaker forms are worth recognising: propext, Quot.sound and Classical.choice are standard and harmless, while sorryAx means the theorem was *admitted rather than proved* — Lean emits an error for the failed tactic but still accepts the declaration, so a theorem can appear in a file, be cited in a table, and not be proved at all. A reader who wants to confirm the present dossier should run the file and compare against the lines above; a reader who wants to confirm that the check is not vacuous should alter one of the defining constants and watch the kernel refuse the result. rhoL is the instructive one: complementation changed from the fixed-point-free involution to a four-cycle leaves eleven of the forty theorems unproved, the pinning theorem among them. A single digit of one row of tableA is the narrow one, and leaves exactly the two theorems about the table.