Emergent Assembly Chirality and Flip Kinematics

Euclidean Stabilizers of the Frame Classes, Enantiotopic Faces of the C_{2h} Core, and Initiator Capping of the Interior Lattice

Verification Code Companion

Charles D. Schaper, Ph.D.
Annals of the Chrysene Formalism, Volume 1, Number 4
Article 91, November 2026

1 Scope and principal findings

This companion supplies executable exact checks for the revised Article 91. It independently recomputes the finite word orbits, the ideal carbon geometry, cap compatibility and signed contact model. The verifier executes **108,715 explicit predicates** in each of normal and optimized Python modes and rejects ten deliberate false controls. All checks use the Python standard library; rational geometry uses `fractions.Fraction`.

The inherited verifier did not pass as supplied. The corrected four-cell coordinate is $(1, -2)$ rather than $(1, 2)$, and the supplied eighteen-carbon coordinates give

$$M = \frac{1}{8}\begin{pmatrix} 177 & -91\sqrt{3} \\ -91\sqrt{3} & 507 \end{pmatrix}, \qquad \lambda_{\pm} = \frac{171 \pm \sqrt{13017}}{4}.$$

These values replace the incompatible inherited covariance constants. The revised verifier also implements checks that the original appendix described but did not execute, notably continuous-reflection candidates, complete finite cap operations and thermal energy exchange.

2 Coverage by verification block

Z1 Ideal carbon geometry.
The corrected four hexagons reconstruct exactly the eighteen sites. Half-turn closure, all twelve discrete polyhex operations and all continuous planar reflection candidates are checked with exact arithmetic. Covariance entries and the characteristic discriminant are recomputed.

Z2 Site words and frame classes.
All site words for $2 \le l \le 14$ are enumerated. Complement invariance, reversal equivariance, two lifts per field and size-two versus size-four frame orbits are checked. This finite sweep corroborates the general combinatorial proof in the manuscript; it does not replace it.

Z3 Distinguished lengths.
Integer formulas are evaluated at $l = 6, 12, 16, 54$. The $l = 54$ fraction is exactly 2^{-26}. The long lengths are formula evaluations, not exhaustive molecular or word enumeration.

Z4 Decorated cap model.
All four flip/reversal operations are tested on one-end-capped decorated stacks of lengths two through eight, with either D or T as the other species. These are formal candidate operations under the manuscript's realization hypothesis, not a search over molecular conformations.

Z5 Resource and boundary arithmetic.
Rational supply, surplus and thresholds are checked. The two lateral boundary orientations fit

the required locants; the strict interior fails. This distinguishes sufficient stock from a feasible saturated placement.

Z6 Sign-model stabilizers.
All four D and sixteen T sign configurations are tested under identity, half-turn, horizontal sign reversal and inversion. Counts are 2/2 and 4/4/8 for the stated reduced sign model.

Z7 Contact bias and thermal invariance.
The exact gap $E_1 - E_0 = 4n_{\mathrm{anti}}\epsilon$ is checked with ϵ factored out. Every second-factor half-turn orbit has identical energy multisets for the two registries. A correlated anti-diagonal ensemble has symmetric one-body marginals but unequal registry energy histograms, refuting the original weaker averaging claim.

3 Completeness of the finite reflection test

Let V contain centered integer pairs (u, v) representing $(u/4, \sqrt{3}v/4)$. A symmetry fixes the centroid. Fix $p = (-7, 1)$; any planar reflection maps p to one of the four points of V having norm $u^2+3v^2 = 52$. For $q \neq -p$, its axis is spanned by $p + q$; for $q = -p$, the axis is perpendicular to p. These formulas determine the unique reflection in either case. The script tests all four candidates and supplies an exact failure witness for each. Thus the finite check plus this geometric reduction excludes every planar reflection, including off-lattice axes.

4 Reproduction and evidence

Run `python verify_a91.py` and `python -O verify_a91.py` from the source directory. Both commands must print the same predicate count and write identical `verification_values_a91.tex` and `verification_report.json`. Explicit `check` calls remain enabled in optimized mode. The standalone LaTeX source extracts the same Python file through `filecontents*`; typesetting does not execute Python.

The inherited source parameters $p_3 = 4.88 \times 10^{-4}$, $\eta_{\mathrm{total}} = 0.998437$, $R_D = 42354$, $R_T = 208$ and $q = 10$ are imported illustrative inputs. They are not measured yields verified by this code. Decimal displays are rounded presentations of model arithmetic where applicable.

5 Limits of the computational evidence

No finite certificate establishes a chemically faithful decorated realization, a synthetic route, molecular chirality, equilibrium sampling, a kinetic barrier or biological function. The general Euclidean and combinatorial arguments remain mathematical manuscript proofs. The corrected thermal conclusion requires invariance of the *joint* conformer weight under a half-turn of the second partner, which is stronger than symmetry of both one-body marginals.

6 Complete executable source

verify_a91.py

```
#!/usr/bin/env python3
"""Article 91 exact model verifier. Python standard library only.
Assertions use explicit checks and remain active under Python -O.
"""
CHECKS = 0
def check(condition, message):
    global CHECKS
    CHECKS += 1
    if not condition:
        raise AssertionError(message)
"""
verify_a91.py
Verification script for Article 91: "Emergent Assembly Chirality and Flip Kinematics"
Annals of the Chrysene Formalism, Vol. 1, No. 4 (2026) -- Final Revised Edition.

Uses Python standard library only (fractions.Fraction, itertools, math).
Verifies Blocks Z1--Z7 and writes verification_values_a91.tex.
"""
from fractions import Fraction
import itertools
macros = {}

def emit(name, val):
    check(name not in macros, f'Duplicate macro: {name}')
    macros[name] = str(val)
POLYHEXES = {1: [(0, 0), (0, 1), (0, 2), (0, 3)], 2: [(0, 0), (0, 1), (0, 2), (1, -1)], 3: [(0, 0), (0, 1), (0, 2), (1, 0)], 4: [(0, 0), (0, 1), (1, -2), (1, -1)], 5: [(0, 0), (0, 1), (1, 1), (1, 0)], 6: [(0, 0), (0, 1), (1, -1), (1, 1)], 7: [(0, 0), (1, -2), (1, -1), (2, -1)]}

def rot_cell(q, r):
    return (-r, q + r)

def refl_cell(q, r):
    return (q, -q - r)

def apply_d6(cells, k_rot, do_refl):
    pts = list(cells)
    if do_refl:
        pts = [refl_cell(q, r) for q, r in pts]
    for _ in range(k_rot):
        pts = [rot_cell(q, r) for q, r in pts]
    min_q, min_r = min(pts)
    return tuple(sorted(((q - min_q, r - min_r) for q, r in pts)))
S4_norm = apply_d6(POLYHEXES[4], 0, False)
s4_rot_stabilizer = [k for k in range(6) if apply_d6(POLYHEXES[4], k, False) == S4_norm]
s4_refl_stabilizer = [k for k in range(6) if apply_d6(POLYHEXES[4], k, True) == S4_norm]
check(s4_rot_stabilizer == [0, 3], 'Gamma_4^px rotations must be {I, R^3}')
check(s4_refl_stabilizer == [], 'Chrysene polyhex S_4 must have 0 planar reflections')
PERIM_COORDS = {1: (-2, 2), 2: (-1, 3), 3: (1, 3), 4: (2, 2), 5: (4, 0), 6: (5, -1), 7: (5, -3), 8: (4, -4), 9: (2, -4), 10: (1, -3), 11: (-1, -1), 12: (-2, 0)}
FUSION_COORDS = {1: (-1, 1), 2: (1, -1), 3: (1, 1), 4: (2, -2), 5: (2, 0), 6: (4, -2)}
ALL_18 = list(PERIM_COORDS.values()) + list(FUSION_COORDS.values())
check(len(ALL_18) == 18, 'predicate at source line 63')
mean_X = Fraction(sum((X for X, _ in ALL_18)), 18)
mean_Y = Fraction(sum((Y for _, Y in ALL_18)), 18)
check((mean_X, mean_Y) == (Fraction(3, 2), Fraction(-1, 2)), 'predicate at source line 68')
I_xx = sum(((Fraction(X, 2) - Fraction(mean_X, 2)) ** 2 for X, _ in ALL_18))
I_yy = Fraction(3, 1) * sum(((Fraction(Y, 2) - Fraction(mean_Y, 2)) ** 2 for _, Y in ALL_18))
I_xy_coeff = sum(((Fraction(X, 2) - Fraction(mean_X, 2)) * (Fraction(Y, 2) - Fraction(mean_Y, 2)) for X, Y in ALL_18))
check(I_xx == Fraction(177, 8), 'predicate at source line 74')
check(I_yy == Fraction(507, 8), 'predicate at source line 75')
check(I_xy_coeff == Fraction(-91, 8), 'predicate at source line 76')
```

```
IOTA_FULL = {1: 7, 7: 1, 2: 8, 8: 2, 3: 9, 9: 3, 4: 10, 10: 4, 5: 11, 11: 5, 6: 12, 12: 6}
for j, (X, Y) in PERIM_COORDS.items():
    check((3 - X, -1 - Y) == PERIM_COORDS[IOTA_FULL[j]], 'predicate at source line 80')
emit('valPolyhexCount', len(POLYHEXES))
emit('valChrysenePlanarOrder', len(s4_rot_stabilizer) + len(s4_refl_stabilizer))
emit('valChryseneReflections', len(s4_refl_stabilizer))
emit('valCtwohOrder', 4)
emit('valIxx', f'{I_xx.numerator}/{I_xx.denominator}')
emit('valIyy', f'{I_yy.numerator}/{I_yy.denominator}')
emit('valIxyCoeff', f'{I_xy_coeff.numerator}/{I_xy_coeff.denominator}')

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

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

def agr_word(w):
    return tuple(((w[i] + w[i + 1]) % 2 for i in range(len(w) - 1)))
for l in range(2, 15):
    words = list(itertools.product((0, 1), repeat=l))
    for w in words:
        eta = agr_word(w)
        eta_rev = tuple(reversed(eta))
        check(agr_word(rev_word(w)) == eta_rev, 'predicate at source line 107')
        check(agr_word(flip_word(rev_word(w))) == eta_rev, 'predicate at source line 108')
    seen = set()
    size2_orbits, size4_orbits = (0, 0)
    for w in words:
        orb = frozenset([w, flip_word(w), rev_word(w), flip_word(rev_word(w))])
        if orb not in seen:
            seen.add(orb)
            eta = agr_word(w)
            is_pal = eta == tuple(reversed(eta))
            if len(orb) == 2:
                check(is_pal, 'Size-2 Fr-orbit must be a palindromic (achiral) field')
                size2_orbits += 1
            elif len(orb) == 4:
                check(not is_pal, 'Size-4 Fr-orbit must be a non-palindromic (chiral) field')
                size4_orbits += 1
    n_ach_f = 2 ** (l // 2)
    n_chi_f = (2 ** (l - 1) - n_ach_f) // 2
    check((size2_orbits, size4_orbits) == (n_ach_f, n_chi_f), 'predicate at source line 126')

def counts_at_l(l):
    n_ach = 2 ** (l // 2)
    n_chi = (2 ** (l - 1) - n_ach) // 2
    return (n_ach + n_chi, n_ach, n_chi)
tot6, ach6, chi6 = counts_at_l(6)
tot12, ach12, chi12 = counts_at_l(12)
tot16, ach16, chi16 = counts_at_l(16)
_, ach54, _ = counts_at_l(54)
emit('valWordsSix', 64)
emit('valFieldsSix', 32)
emit('valClassesSix', tot6)
emit('valAchiralClassesSix', ach6)
emit('valChiralClassesSix', chi6)
emit('valAchiralFieldsSix', ach6)
emit('valChiralFieldsSix', 2 * chi6)
emit('valChiralPctSix', f'{100.0 * (2 * chi6) / 32.0:.1f}')
emit('valClassesTwelve', f'{tot12:,}')
emit('valAchiralClassesTwelve', ach12)
emit('valChiralClassesTwelve', chi12)
emit('valChiralPctTwelve', f'{100.0 * (2 * chi12) / 2 ** 11:.3f}')
emit('valClassesSixteen', f'{tot16:,}')
emit('valAchiralClassesSixteen', ach16)
emit('valChiralClassesSixteen', f'{chi16:,}')
emit('valAchiralFieldsFiftyFour', f'{ach54:,}')
```

```
emit('valAchiralFracFiftyFour', '1.49 \\times 10^{-8}')
I_SET = frozenset([3, 6, 12])
IOTA_PRI = {3: 9, 9: 3, 6: 12, 12: 6}
I_FLIP_SET = frozenset((IOTA_PRI[loc] for loc in I_SET))
check(I_FLIP_SET == frozenset([6, 9, 12]) != I_SET, 'predicate at source line 162')
R_D, R_T, R_I = (Fraction(42354, 1), Fraction(208, 1), Fraction(1, 1))
rho_star = R_D / R_T
q_part = Fraction(10, 1)
x_base = Fraction(1, 10)
coop_x01 = q_part * rho_star / (x_base * (1 + x_base))
check(round(float(coop_x01)) == 18511, 'predicate at source line 169')
init_prod = R_T * x_base
init_surplus = init_prod - R_I
x_min_1 = Fraction(2, 208)
x_min_9 = Fraction(10, 208)
x_min_18 = Fraction(19, 208)
nx, ny = (3, 3)
kx2_cols, kx0_cols = (0, 0)
for ky in range(ny):
    req_kx2 = {3} | ({12} if ky > 0 else set()) | ({6} if ky < ny - 1 else set())
    req_kx0 = {9} | ({12} if ky > 0 else set()) | ({6} if ky < ny - 1 else set())
    check(req_kx2.issubset(I_SET), 'predicate at source line 182')
    check(req_kx0.issubset(I_FLIP_SET), 'predicate at source line 183')
    kx2_cols += 1
    kx0_cols += 1
check((kx2_cols, kx2_cols + kx0_cols) == (3, 6), 'predicate at source line 186')
emit('valPThree', '4.88 \\times 10^{-4}')
emit('valEtaTotal', '0.998437')
emit('valCoopXPointOne', f'{round(float(coop_x01)):,}')
emit('valInitiatorProduced', f'{float(init_prod):.1f}')
emit('valInitiatorDemand', int(R_I))
emit('valInitiatorSurplus', f'{float(init_surplus):.1f}')
emit('valXMinOneCap', f'{float(x_min_1):.5f}')
emit('valXMinNineCaps', f'{float(x_min_9):.5f}')
emit('valXMinEighteenCaps', f'{float(x_min_18):.5f}')
emit('valBoxColumns', nx * ny)
emit('valBoxXBoundaryCols', kx2_cols)
emit('valBoxDualXBoundaryCols', kx2_cols + kx0_cols)

def classify_sign_states(locants, orbits):
    states = list(itertools.product((-1, 1), repeat=len(locants)))
    c2_cnt, ci_cnt, c1_cnt = (0, 0, 0)
    for st in states:
        d = dict(zip(locants, st))
        n_anti = sum((1 for a, b in orbits if d[a] == -d[b]))
        E0 = -len(locants)
        E1 = -sum((d[loc] * d[IOTA_PRI[loc]] for loc in locants))
        check(E1 - E0 == 4 * n_anti, 'predicate at source line 212')
        if n_anti == 0:
            c2_cnt += 1
        elif n_anti == len(orbits):
            ci_cnt += 1
        else:
            c1_cnt += 1
    return (len(states), c2_cnt, ci_cnt, c1_cnt)
totD, c2D, ciD, c1D = classify_sign_states((6, 12), [(6, 12)])
totT, c2T, ciT, c1T = classify_sign_states((3, 6, 9, 12), [(3, 9), (6, 12)])
check((totD, c2D, ciD, c1D) == (4, 2, 2, 0), 'predicate at source line 223')
check((totT, c2T, ciT, c1T) == (16, 4, 4, 8), 'predicate at source line 224')
emit('valSignStatesD', totD)
emit('valCiStatesD', ciD)
emit('valCtwoStatesD', c2D)
emit('valSignStatesT', totT)
emit('valCiStatesT', ciT)
emit('valCtwoStatesT', c2T)
emit('valConeStatesT', c1T)
CENTERS = [(0, 0), (0, 2), (3, -3), (3, -1)]
OFFSETS = [(2, 0), (1, 1), (-1, 1), (-2, 0), (-1, -1), (1, -1)]
```

```
check({(x + a, y + b) for x, y in CENTERS for a, b in OFFSETS} == set(ALL_18), 'predicate at source line 237')
check({(3 * q, q + 2 * r) for q, r in POLYHEXES[4]} == set(CENTERS), 'predicate at source line 238')
check({(3 - x, -1 - y) for x, y in ALL_18} == set(ALL_18), 'predicate at source line 239')
V = {(2 * x - 3, 2 * y + 1) for x, y in ALL_18}

def dot(a, b):
    return a[0] * b[0] + 3 * a[1] * b[1]
p = (-7, 1)
images = sorted((q for q in V if dot(q, q) == dot(p, p)))
check(images == [(-7, 1), (-5, 3), (5, -3), (7, -1)], 'predicate at source line 245')

def reflected(v, q):
    a = (p[0] + q[0], p[1] + q[1])
    if a == (0, 0):
        return tuple((v[i] - 2 * Fraction(dot(p, v), dot(p, p)) * p[i] for i in range(2)))
    return tuple((2 * Fraction(dot(a, v), dot(a, a)) * a[i] - v[i] for i in range(2)))
reflection_witnesses = []
for q in images:
    check(reflected(p, q) == q, 'predicate at source line 253')
    check(dot(reflected((1, 0), q), reflected((0, 1), q)) == 0, 'predicate at source line 254')
    check(dot(reflected((1, 0), q), reflected((1, 0), q)) == 1, 'predicate at source line 255')
    check(dot(reflected((0, 1), q), reflected((0, 1), q)) == 3, 'predicate at source line 256')
    check({reflected(v, q) for v in V} != V, 'predicate at source line 257')
    witness = next((v for v in sorted(V) if reflected(v, q) not in V))
    reflection_witnesses.append([list(q), list(witness), list(map(str, reflected(witness, q)))])
check(I_xx + I_yy == Fraction(171, 2), 'predicate at source line 262')
check((I_xx - I_yy) ** 2 / 4 + 3 * I_xy_coeff ** 2 == Fraction(13017, 16), 'predicate at source line 263')
for length in range(2, 15):
    fibres = {}
    for w in itertools.product((0, 1), repeat=length):
        fibres.setdefault(agr_word(w), []).append(w)
    check(len(fibres) == 2 ** (length - 1), 'predicate at source line 269')
    for field, lifts in fibres.items():
        check(len(lifts) == 2 and flip_word(lifts[0]) == lifts[1], 'predicate at source line 271')
        check(agr_word(flip_word(lifts[0])) == field, 'predicate at source line 272')
check((tot6, ach6, chi6) == (20, 8, 12), 'predicate at source line 274')
check((tot12, ach12, chi12) == (1056, 64, 992), 'predicate at source line 275')
check((tot16, ach16, chi16) == (16512, 256, 16256), 'predicate at source line 276')
check(Fraction(ach54, 2 ** 53) == Fraction(1, 2 ** 26), 'predicate at source line 277')
check(Fraction(2 * chi6, 32) == Fraction(3, 4), 'predicate at source line 278')

def transform_stack(stack, flip, rev):
    answer = list(stack[::-1] if rev else stack)
    if flip:
        answer = [(species, frozenset((IOTA_PRI[x] for x in sites)), 1 - bit) for species, sites, bit in answer]
    return tuple(answer)
D = frozenset([6, 12])
T = frozenset([3, 6, 9, 12])
for length in range(2, 9):
    for terminal, sites in [('D', D), ('T', T)]:
        for w in itertools.product((0, 1), repeat=length):
            stack = (('I', I_SET, w[0]),) + tuple(((terminal, sites, b) for b in w[1:]))
            stabilizer = [(f, r) for f in (False, True) for r in (False, True) if transform_stack(stack, f, r) == stack]
            check(stabilizer == [(False, False)], 'predicate at source line 293')
check(init_prod == Fraction(104, 5) and init_surplus == Fraction(99, 5), 'predicate at source line 295')
for caps, threshold in [(1, x_min_1), (9, x_min_9), (18, x_min_18)]:
    check(208 * threshold - 1 == caps, 'predicate at source line 297')
check(not T.issubset(I_SET) and (not T.issubset(I_FLIP_SET)), 'predicate at source line 298')

def c2(z):
    return tuple((z[i ^ 1] for i in range(len(z))))

def neg(z):
    return tuple((-a for a in z))
```

```
def energy(a, b, eta):
    other = c2(b) if eta else b
    return -sum((x * y for x, y in zip(a, other)))
from collections import Counter
sign_results = {}
for size in (2, 4):
    states = list(itertools.product((-1, 1), repeat=size))
    totals = Counter()
    for z in states:
        stab = tuple((name for name, v in [('E', z), ('C2', c2(z)), ('H', neg(z)), ('Inv', neg(c2(z)))] if v == z))
        check(stab in [('E', 'C2'), ('E', 'Inv'), ('E',)], 'predicate at source line 312')
        totals[stab] += 1
        anti = sum((z[i] != z[i + 1] for i in range(0, size, 2)))
        check(energy(z, z, 1) - energy(z, z, 0) == 4 * anti, 'predicate at source line 315')
        for a in states:
            check(energy(a, c2(z), 0) == energy(a, z, 1), 'predicate at source line 317')
            check(energy(a, c2(z), 1) == energy(a, z, 0), 'predicate at source line 318')
            partner_orbit = set([z, c2(z)])
            check(Counter((energy(a, b, 0) for b in partner_orbit)) == Counter((energy(a, b, 1) for b in partner_orbit)), 'predicate at source line 321')
    expected = {('E', 'C2'): 2 ** (size // 2), ('E', 'Inv'): 2 ** (size // 2)}
    if size == 4:
        expected['E',] = 8
    check(dict(totals) == expected, 'predicate at source line 324')
    sign_results[str(size)] = {','.join(k): v for k, v in totals.items()}
anti = (-1, 1)
anti2 = c2(anti)
diagonal = [(anti, anti), (anti2, anti2)]
h0 = Counter((energy(a, b, 0) for a, b in diagonal))
h1 = Counter((energy(a, b, 1) for a, b in diagonal))
check(h0 == {-2: 2} and h1 == {2: 2} and (h0 != h1), 'predicate at source line 331')
check(Counter((a for a, b in diagonal)) == Counter((c2(a) for a, b in diagonal)), 'predicate at source line 332')
check(Counter((b for a, b in diagonal)) == Counter((c2(b) for a, b in diagonal)), 'predicate at source line 333')
negative_controls = [('original covariance', I_xx == Fraction(159, 2)), ('original covariance yy', I_yy == Fraction(201, 2)), ('original covariance xy', I_xy_coeff == Fraction(-57, 2)), ('incorrect six-class count', tot6 == 19), ('incorrect palindrome count', ach6 == 7), ('incorrect 54 ratio', Fraction(ach54, 2 ** 53) == Fraction(1, 2 ** 27)), ('cap occupies interior', T.issubset(I_SET)), ('marginals imply partition equality', h0 == h1), ('C2 anti state is fixed', c2(anti) == anti), ('reflection preserves carbon set', {reflected(v, images[0]) for v in V} == V)]
for label, wrong in negative_controls:
    check(not wrong, 'Mutation survived: ' + label)
with open('verification_values_a91.tex', 'w', encoding='utf-8') as f:
    f.write('% Generated by the Article 91 revised exact verifier.\n')
    for name, value in macros.items():
        f.write('\\newcommand{\\' + name + '}{' + value + '}\n')
import json
report = {'checks': CHECKS, 'negative_controls': len(negative_controls), 'enumerated_lengths': [2, 14], 'covariance': ['177/8', '507/8', '-91*sqrt(3)/8'], 'reflection_witnesses': reflection_witnesses, 'sign_stabilizers': sign_results, 'imported_not_validated': ['p3=4.88e-4', 'eta_total=0.998437', 'R_D=42354', 'R_T=208', 'q=10'], 'scope': 'Finite arithmetic, supplied ideal geometry, abstract decorated-frame and sign models. No molecular realization or synthesis validation.'}
with open('verification_report.json', 'w') as f:
    json.dump(report, f, indent=2)
print(json.dumps({'result': 'PASS', 'checks': CHECKS, 'negative_controls': len(negative_controls)}, sort_keys=True))
```