Executable Verification

Symmetry-Compatible Information in the Chrysene Formalism

Axiomatic Realization, Coding Obstructions, Constructive Repair, and Falsifiable Molecular Models

Charles D. Schaper, Ph.D.

Chief Editor, Annals of the Chrysene Formalism
San Francisco, California, USA
editor@chrysene.com

Scope. This executable supplement independently evaluates the finite algebra, coding construction, decoder, quotient, and channel-count claims of Article 100. It supplies 77,219 exact checks, including ten deliberately false controls. It is not a proof of a molecular realization, biogenesis, or uniqueness of a chemical substrate.

1 Representation and reproducibility

The field is $\mathbb{F}_2[w]/(w^2 + w + 1)$, with integers $0, 1, 2, 3$ representing $0, 1, w, w^2$. Addition is bitwise XOR; multiplication is an explicit table. All 1,024 words are generated rather than sampled. Only the Python standard library is required. Run `python3 verify_a100.py`; the result is JSON on standard output. Also run `python3 -O verify_a100.py`. Checks use explicit exceptions rather than removable assertions. Both runs must return the same report and exit successfully.

2 Coverage and limits

The original and repaired parity-check matrices are

$$H = \begin{pmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & w & w^2 \end{pmatrix}, \qquad K = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & w & w^2 \end{pmatrix}.$$

The program verifies ranks, the original four-word reversal intersection, and a distance-one obstruction between distinct reversal orbits. The repaired code has 64 words, minimum distance three, 16 reversal-fixed words, and 40 message orbits. Its ambient quotient has 544 orbits. Direct syndrome decoding is checked on every received word, with equivariance, all 1,600 quotient pairs, and the complete quotient-ball partition. Sixteen balls have size ten and 24 have size sixteen.

All 65,536 transmitted-codeword/received-word pairs are tested: decoder orbit success equals membership in the union of the radius-one balls about a word and its reversal. The resulting distance-shell vectors are $(1, 15, 0, 0, 0, 0)$ for fixed words and $(1, 15, 0, 4, 9, 3)$ for nonfixed words. Rational evaluations at $p = 0, 1/100, \ldots, 1$ independently agree with the manuscript formulas for both priors. These evaluations do not prove an identity for all real p; that identity follows analytically from the shell counts and channel model in the manuscript. The induced entropy $21/4$ is checked by rational arithmetic using the known dyadic orbit probabilities.

The program enumerates all 180 invertible two-by-two field matrices and their five-point projective actions. The 60 distinct linear actions are even, and the 60 distinct Frobenius-semilinear actions are odd. All 120 projective relabelings of coordinate reversal are excluded from the latter set. This is a finite check of the specified five-point model; the reduction from an arbitrary admissible code to that projective model is a mathematical argument in the manuscript.

Ten negative controls reject wrong counts, matrices, distances, priors, or symmetry claims. They are intentional tests, not failed assertions. The JSON report gives per-category counts. Neither verification language certifies physical feasibility, chemical kinetics, empirical data fit, originality, or the full axiom corpus.

3 Complete executable source

The following listing is generated verbatim from the distributed executable. The executable, rather than PDF text extraction, should be used for reproduction.

```
"""Exact finite verification for Article 100. Python 3, standard library only.
No assert statements: checks remain active under python -O.
"""
from itertools import product, permutations
from collections import Counter
from fractions import Fraction
import json
COUNTS = Counter()
def check(condition, category):
    COUNTS[category] += 1
    if not condition:
        raise RuntimeError("Failed: " + category)
M = ((0,0,0,0),(0,1,2,3),(0,2,3,1),(0,3,1,2))
def mul(a,b): return M[a][b]
def dot(a,b):
    z=0
    for x,y in zip(a,b): z ^= mul(x,y)
    return z
def syn(H,x): return tuple(dot(h,x) for h in H)
def rev(x): return x[::-1]
def frob(x): return tuple(mul(a,a) for a in x)
def canon(x): return min(x,rev(x))
def dist(x,y): return sum(a!=b for a,b in zip(x,y))
def rank(A):
    A=list(map(list,A)); r=0
    for j in range(len(A[0])):
        k=next((k for k in range(r,len(A)) if A[k][j]),None)
        if k is None: continue
        A[r],A[k]=A[k],A[r]
        inv=next(a for a in (1,2,3) if mul(a,A[r][j])==1)
        A[r]=[mul(inv,a) for a in A[r]]
        for k in range(len(A)):
            if k!=r:
                t=A[k][j]; A[k]=[a^mul(t,b) for a,b in zip(A[k],A[r])]
        r+=1
        if r==len(A): break
    return r
H=((1,0,1,1,1),(0,1,1,2,3))
K=((1,1,0,1,1),(0,1,1,2,3))
A=((1,0),(3,1))
W=list(product(range(4),repeat=5))
C=[w for w in W if syn(H,w)==(0,0)]
D=[w for w in W if syn(K,w)==(0,0)]
for a,b,c in product(range(4),repeat=3):
    check(mul(a,b^c)==mul(a,b)^mul(a,c),'field_distributive')
    check(mul(mul(a,b),c)==mul(a,mul(b,c)),'field_associative')
for a,b in product(range(4),repeat=2):
    check(mul(a,b)==mul(b,a),'field_commutative')
for a in range(4):
    check(mul(a,1)==a and a^a==0 and mul(mul(a,a),mul(a,a))==a,'field_identity_frobenius')
for a in (1,2,3): check(any(mul(a,b)==1 for b in (1,2,3)),'field_inverse')
check(rank(H)==rank(K)==2 and rank(H+tuple(map(rev,H)))==4,'ranks')
check(len(C)==len(D)==64,'code_counts')
check(set(C)&set(map(rev,C))=={tuple(mul(a,b) for b in (1,2,2,2,1)) for a in range(4)},'intersection')
x=(0,0,1,2,3); y=(0,2,1,2,3)
check(x in C and rev(y) in C and canon(x)!=canon(y) and min(dist(x,y),dist(x,rev(y)))==1,'obstruction_witness')
check({(a^mul(3,b)^mul(2,c),a^mul(2,b)^mul(3,c),a,b,c) for a,b,c in product(range(4),repeat=3)}==set(D),'systematic_encoder')
for c in D:
    check(rev(c) in D,'reversal_closure')
    for d in D:
        check(c==d or dist(c,d)>=3,'minimum_distance')
for w in W:
    check(syn(K,rev(w))==tuple(dot(a,syn(K,w)) for a in A),'syndrome_covariance')
E=[(0,)*5]+[tuple(a if j==i else 0 for j in range(5)) for i in range(5) for a in (1,2,3)]
lookup={syn(K,e):e for e in E}
check(len(lookup)==16,'syndrome_bijection')
dec={w:tuple(a^b for a,b in zip(w,lookup[syn(K,w)])) for w in W}
for w in W:
    check(dec[w] in D and dist(w,dec[w])<=1,'syndrome_decoder')
    check(sum(dist(w,c)<=1 for c in D)==1,'unique_nearest')
    check(dec[rev(w)]==rev(dec[w]),'decoder_equivariance')
messages=sorted(set(map(canon,D))); ambient=sorted(set(map(canon,W)))
check(len(messages)==40 and len(ambient)==544,'orbit_counts')
check(sum(rev(c)==c for c in D)==16,'palindromes')
check(len(set(map(canon,C)))==64,'original_orbit_image')
for c,d in product(messages,repeat=2):
    check(c==d or min(dist(c,d),dist(c,rev(d)))>=3,'quotient_distance')
ball_counts=Counter()
for c in messages:
    ball=[y for y in ambient if min(dist(c,y),dist(c,rev(y)))<=1]
    ball_counts[len(ball)]+=1
    for y in ball: check(canon(dec[y])==c,'quotient_ball_decoder')
check(ball_counts=={10:16,16:24},'nonuniform_balls')
```

```
for y in ambient:
    check(sum(min(dist(c,y),dist(c,rev(y)))<=1 for c in messages)==1,'perfect_quotient_partition')
profiles={}
for c in D:
    counts=Counter()
    for y in W:
        k=dist(c,y); success=canon(dec[y])==canon(c)
        union_success=dist(c,y)<=1 or dist(rev(c),y)<=1
        check(success==union_success,'all_pair_orbit_decoding')
        if success: counts[k]+=1
    profile=tuple(counts[k] for k in range(6))
    expected=(1,15,0,0,0,0) if c==rev(c) else (1,15,0,4,9,3)
    check(profile==expected,'success_shell_counts')
    profiles[str(c==rev(c))]=profile
for n in range(101):
    p=Fraction(n,100)
    base=(1-p)**5+5*p*(1-p)**4
    extra=4*(p/3)**3*(1-p)**2+9*(p/3)**4*(1-p)+3*(p/3)**5
    for prior,extra_weight in [('uniform_words',Fraction(3,4)),('uniform_orbits',Fraction(3,5))]:
        direct=Fraction(0)
        seq=D if prior=='uniform_words' else messages
        for c in seq:
            shell=profiles[str(c==rev(c))]
            direct+=sum(Fraction(v)*(p/3)**k*(1-p)**(5-k) for k,v in enumerate(shell))/len(seq)
        check(direct==base+extra_weight*extra,'exact_channel_formula')
check(16*Fraction(6,64)+24*Fraction(5,32)==Fraction(21,4),'induced_entropy')
P=((1,0),(0,1),(1,1),(1,2),(1,3))
def point(v):
    return next(i for i,p in enumerate(P) if any(v==tuple(mul(a,b) for b in p) for a in (1,2,3)))
def parity(p): return sum(p[i]>p[j] for i in range(5) for j in range(i+1,5))%2
linear=set(); semilinear=set()
for a,b,c,d in product(range(4),repeat=4):
    if mul(a,d)^mul(b,c)==0: continue
    B=((a,b),(c,d))
    lp=tuple(point(tuple(dot(r,p) for r in B)) for p in P)
    sp=tuple(point(tuple(dot(r,frob(p)) for r in B)) for p in P)
    check(parity(lp)==0 and parity(sp)==1,'projective_parity')
    linear.add(lp);semilinear.add(sp)
check(len(linear)==len(semilinear)==60 and linear|semilinear==set(permutations(range(5))),'projective_groups')
for lab in permutations(range(5)):
    inv={v:i for i,v in enumerate(lab)}
    r=tuple(lab[4-inv[i]] for i in range(5))
    check(r in linear and r not in semilinear,'all_projective_labelings')
negative={
 'original_reversal_invariant':set(map(rev,C))==set(C),
 'original_quotient_distance_three':min(dist((0,0,1,2,3),(0,2,1,2,3)),dist((0,0,1,2,3),rev((0,2,1,2,3))))>=3,
 'repair_semilinear_invariant':set(map(lambda d:rev(frob(d)),D))==set(D),
 'forty_is_linear_GF4_size':40 in [4**i for i in range(6)],
 'uniform_quotient_balls':len(ball_counts)==1,
 'frobenius_even':all(parity(p)==0 for p in semilinear),
 'nine_palindromes':sum(rev(c)==c for c in D)==9,
 'zero_decoder_valid':all(dist(w,(0,)*5)<=1 for w in W),
 'two_symbol_success':profiles['False'][2]>0,
 'same_prior_weights':Fraction(3,4)==Fraction(3,5)}
for label,value in negative.items(): check(not value,'negative_controls')
print(json.dumps({'status':'passed','checks':sum(COUNTS.values()),'exact_checks':sum(COUNTS.values()),'numerical_checks':0,'
     categories':dict(sorted(COUNTS.items())),'negative_controls':list(negative),'success_shells':profiles,'codewords':64,'
     message_orbits':40,'ambient_orbits':544,'scope':'Exact finite checks, not empirical validation or a proof of general analytic
     hypotheses.'},sort_keys=True,indent=2))
```