Information-Theoretic Mechanics of Survival in Maximum-Entropy Environments

Static Minimax Disturbance Attenuation, Coordination-Deficit Costs, and the Placed Source as Categorical Right Adjoint

Verification Code Companion

Charles D. Schaper, Ph.D.
Annals of the Chrysene Formalism, Volume 1, Number 4
Article 97, November 2026

1 Scope and reproducible result

The revised program executes **64,678 explicit validation calls per mode**: 63,888 exact integer/rational checks and 790 floating-point checks. Ten negative controls are included in that total. Normal and optimized Python produce byte-identical outputs. All checks remain active under `python -O`; the source uses only the Python standard library. A check count is not a count of independent theorems.

The scope comprises bounded combinatorial enumeration, actual support-poset morphisms, two explicit molecular graphs, a stipulated static diagonal attenuation model, and conditional alphabet-capacity arithmetic. It does not establish a chemical synthesis, cosmological mechanism, gravitational entropy bound, dynamic control synthesis, or unique physical variational solution.

2 Inherited artifacts and corrections

The supplied script stops at its gap-ratio assertion. The computed ratio is approximately 1.7634041, so four-decimal rounding gives 1.7634, not 1.7633. The computed chrysene gap is approximately 1.0402776, which rounds to 1.0403. The old code inferred determinants from matching counts rather than independently calculating them, emitted category values without modeling morphisms, and did not test its advertised Stark or categorical claims.

The categorical zero-slack objects are indexed by $3^6 = 729$ species words, not by the 32 registry words in one fibre. The support-poset model has $729\,2^{31}$ expanded objects when every target-containing support is allowed. The revised capacity example consistently uses $L = 233$ and the unrounded radius law; it has 682 interior bits and 4,754 boundary capacity bits, rather than the inherited 660/4440 example.

3 Coverage and exact spectral certificate

V1–V2: slack and additive projection.
Binomial ratios are tested for anchor counts 1–8 and slack 0–39; registry cardinalities for lengths 2–14. The projection from three binary coordinates to two is tested on every pair and fibre. An explicit finite-field multiplication counterexample distinguishes additive projection from a field homomorphism. Three boxes are checked for complete interior valency.

V3: graph invariants.
Both supplied 18-vertex, 21-edge graphs are checked for symmetry and bipartiteness. Perfect matchings are enumerated independently; Bareiss elimination gives determinants -64 and -25. Exact Faddeev–LeVerrier recursion constructs the characteristic polynomial of BB^{T} and checks its Cayley–Hamilton residual. Rational Sturm sequences count zero roots below each claimed first-root interval and one inside it. Thus

$$1.0402 < g_c/\beta < 1.0403, \qquad 0.5899 < g_t/\beta < 0.5900.$$

The determinant check excludes a zero singular value. Bipartite block structure converts the smallest positive eigenvalue of BB^{T} to the HOMO–LUMO gap. The ordering follows from these exact disjoint intervals, not the decimal Jacobi calculation. This is an unperturbed-graph result.

V4: static surrogate.
Rational weighted-norm inequalities and coordinate attainment are checked for up to four controls, with integer budgets and test disturbances. Floating evaluations check logistic gate inversion and positive residual entropy at finite bias. These support the manuscript's static diagonal-operator theorem, not a general dynamic $\mathcal{H}_\infty$ claim. All 4^4 plaquette labels satisfy the telescoping identity.

V5–V6: categories and feasibility.
Every object and composable triple of the four-registry-state, two-species support poset is tested; hom-set equivalence, unit and counit are checked. The separate Lean file proves the support-poset adjunction with arbitrary species and supports. The three-component indicator condition is tested on all eight Boolean triples; it is not a uniqueness theorem for geometries.

V7: capacity.
Rational square-root brackets and a stated rational enclosure of π certify the tile floor at $L = 233$. Exact inequalities validate constants for an alternative sufficient proof of the manuscript's all-L capacity inequality. The asymptotic decimal is numerical. The bound assumes the prescribed radius, alphabets and fixed 54-node scaffold; it is not a Bekenstein–Hawking law.

The squared-spectrum polynomials, in descending coefficient order, are

$$\begin{aligned} p_c(t) &= t^9 - 21t^8 + 180t^7 - 824t^6 + 2214t^5 \\ &\quad - 3605t^4 + 3533t^3 - 1990t^2 + 577t - 64, \\ p_t(t) &= t^9 - 21t^8 + 180t^7 - 822t^6 + 2192t^5 \\ &\quad - 3510t^4 + 3321t^3 - 1731t^2 + 415t - 25. \end{aligned}$$

For the radius estimate, π is enclosed between 3.141592653589793238 and 3.141592653589793239, and the square-root bracket is computed by integer square root at denominator 10^{12}. The resulting floor is 2,377 tiles. This uses the standard analytic enclosure of π as input. For an independent all-length certificate, $L \geq 233$ and $(c - 0.63)^2(233) > 16$ imply $R(L) > 0.63\sqrt{L}$. Since $\pi > 3$ and $a = 1/2$, the boundary capacity exceeds $2(24(0.63)^2L - 1)$. The exact slope and margin checks make this strictly larger than $2L + 216$ throughout that half-line.

4 Reproduction and negative controls

Run `python run_verification.py`. The runner checks the pinned source, executes both modes in fresh directories, and compares every output byte. JSON results and manuscript macros are produced by the executed code. Ten false alternatives are explicitly rejected: old gap ratio, incorrect determinant, field-multiplicativity, wrong species count, reversed support morphism, cross-species morphism, zero finite-bias entropy, missing feasible component, old interior bits and old tile count. They are negative assertion tests, not independently randomized experiments.

5 Complete reproducible source

Article 97 source

```
#!/usr/bin/env python3
"""Article 97 revised finite verification. Python standard library only.
Explicit checks survive -O. Exact arithmetic and floating checks are counted
separately; negative controls are included. No chemical/cosmological validation.
"""
from fractions import Fraction as Q
from itertools import product, permutations
from collections import Counter
from pathlib import Path
import math, json, hashlib
COUNTS=Counter()
def check(ok, block, numerical=False):
    COUNTS[block]+=1
    COUNTS['numerical' if numerical else 'exact']+=1
    if not ok: raise RuntimeError('FAILED: '+block)
def negative(ok):
    check(not ok,'negative_controls')

def matmul(a,b):
    return [[sum(x*y for x,y in zip(row,col)) for col in zip(*b)] for row in a]
def charpoly(a):
    n=len(a); b=[[int(i==j) for j in range(n)] for i in range(n)]; out=[1]
    for k in range(1,n+1):
        ab=matmul(a,b); tr=sum(ab[i][i] for i in range(n))
        check(tr%k==0,'charpoly_integrality')
        c=-tr//k;out.append(c)
        b=[[ab[i][j]+c*int(i==j) for j in range(n)] for i in range(n)]
    check(all(x==0 for row in b for x in row),'cayley_hamilton')
    return out
def bareiss(a):
    a=[row[:] for row in a]; sign=1;prev=1;n=len(a)
    for k in range(n-1):
        if a[k][k]==0:
            r=next((r for r in range(k+1,n) if a[r][k]),None)
            if r is None:return 0
            a[k],a[r]=a[r],a[k];sign=-sign
        pivot=a[k][k]
        for i in range(k+1,n):
            for j in range(k+1,n):
                v=a[i][j]*pivot-a[i][k]*a[k][j]
                check(v%prev==0,'bareiss_integrality');a[i][j]=v//prev
            a[i][k]=0
        prev=pivot
    return sign*a[-1][-1]
def trim(p):
    while len(p)>1 and p[0]==0:p=p[1:]
    return p
def prem(p,q):
    p=list(map(Q,p));q=list(map(Q,q))
    while len(p)>=len(q) and any(p):
        v=p[0]/q[0]
        p=trim([p[i]-(v*q[i] if i<len(q) else 0) for i in range(len(p))])
    return p
def sturm(p):
    seq=[list(map(Q,p)),[Q(c*(len(p)-1-i)) for i,c in enumerate(p[:-1])]]
    while True:
        r=[-x for x in prem(seq[-2],seq[-1])]
        if not any(r):break
        seq.append(r)
    return seq
def peval(p,x):
    r=Q(0)
    for c in p:r=r*x+c
```

```
    return r
def variations(seq,x):
    vals=[peval(p,x) for p in seq]; signs=[1 if v>0 else -1 for v in vals if v]
    return sum(a!=b for a,b in zip(signs,signs[1:]))
def rootcount(seq,a,b):return variations(seq,a)-variations(seq,b)

V_A_chr = ["v1", "v3", "v5", "v8", "v10", "v12", "w2", "w3", "w6"]
V_B_chr = ["v2", "v4", "v6", "v7", "v9", "v11", "w1", "w4", "w5"]
V_CHR = V_A_chr + V_B_chr
IDX_CHR = {v: i for i, v in enumerate(V_CHR)}
ADJ_CHR = {
    "v1": ["v2", "w1"], "v2": ["v1", "v3"], "v3": ["v2", "v4"], "v4": ["v3", "w3"],
    "v5": ["w5", "v6"], "v6": ["v5", "w6"], "v7": ["w6", "v8"], "v8": ["v7", "v9"],
    "v9": ["v8", "v10"], "v10": ["v9", "w4"], "v11": ["w2", "v12"], "v12": ["v11", "w1"
        ],
    "w1": ["v12", "v1", "w3"], "w2": ["v11", "w4", "w5"], "w3": ["v4", "w1", "w5"],
    "w4": ["v10", "w2", "w6"], "w5": ["v5", "w2", "w3"], "w6": ["v6", "v7", "w4"],
}
A_chr = [[0]*18 for _ in range(18)]
for u, nbrs in ADJ_CHR.items():
    for v in nbrs:
        A_chr[IDX_CHR[u]][IDX_CHR[v]] = 1

V_A_tet = [f"u{2*k}" for k in range(5)] + [f"v{2*k+1}" for k in range(4)]
V_B_tet = [f"v{2*k}" for k in range(5)] + [f"u{2*k+1}" for k in range(4)]
V_TET = V_A_tet + V_B_tet
IDX_TET = {v: i for i, v in enumerate(V_TET)}
A_tet = [[0]*18 for _ in range(18)]
def add_tet_edge(a, b):
    A_tet[IDX_TET[a]][IDX_TET[b]] = 1
    A_tet[IDX_TET[b]][IDX_TET[a]] = 1

for k in range(8):
    add_tet_edge(f"u{k}", f"u{k+1}")
    add_tet_edge(f"v{k}", f"v{k+1}")
for k in (0, 2, 4, 6, 8):
    add_tet_edge(f"u{k}", f"v{k}")

def jacobi_evals(M_in, tol=1e-13):
    n = len(M_in)
    A = [[float(M_in[r][c]) for c in range(n)] for r in range(n)]
    while True:
        max_val, p, q = 0.0, 0, 1
        for r in range(n):
            for c in range(r + 1, n):
                if abs(A[r][c]) > max_val:
                    max_val, p, q = abs(A[r][c]), r, c
        if max_val < tol:
            break
        theta = 0.5 * math.atan2(2.0 * A[p][q], A[q][q] - A[p][p])
        cs, sn = math.cos(theta), math.sin(theta)
        app, aqq, apq = A[p][p], A[q][q], A[p][q]
        A[p][p] = cs*cs*app - 2.0*sn*cs*apq + sn*sn*aqq
        A[q][q] = sn*sn*app + 2.0*sn*cs*apq + cs*cs*aqq
        A[p][q] = A[q][p] = 0.0
        for k in range(n):
            if k != p and k != q:
                akp, akq = A[k][p], A[k][q]
                A[k][p] = A[p][k] = cs*akp - sn*akq
                A[k][q] = A[q][k] = sn*akp + cs*akq
    return sorted(A[i][i] for i in range(n))


# V1: positive combinatorial counts and explicitly uniform registry model.
for nf in range(1,9):
    check(math.comb(nf,nf)==1,'slack')
    for g in range(40):
        a=math.comb(nf+g,nf);b=math.comb(nf+g+1,nf)
```

```
        check(b>a,'slack');check(b*(g+1)==a*(nf+g+1),'slack')
for l in range(2,15):
    check(math.comb(1+2**(l-1)-1,1)==2**(l-1),'registry')
# V2: additive projection, not a field homomorphism; strain surrogate.
for u,v in product(range(8),repeat=2):
    check(((u^v)&3)==((u&3)^(v&3)),'projection')
for y in range(4):check(sum((x&3)==y for x in range(8))==2,'projection')
for L in range(1,31):
    check(6*L**3-4*L**2==2*L**2*(3*L-2)>0,'strain')
# Explicit additive-but-not-multiplicative field projection (binary bases).
def fmul(a,b,mod,degree):
    r=0
    while b:
        if b&1:r^=a
        b>>=1;a<<=1
        if a&(1<<degree):a^=mod
    return r
check((fmul(2,2,11,3)&3)!=fmul(2,2,7,2),'field_counterexample')
for dims in [(3,3,3),(3,3,6),(4,4,4)]:
    verts=set(product(*(range(d) for d in dims)))
    for v in verts:
        deg=sum(tuple(v[j]+(s if j==i else 0) for j in range(3)) in verts for i in
            range(3) for s in [-1,1])
        if all(0<v[i]<dims[i]-1 for i in range(3)):check(deg==6,'interior_valency')
# V3: independent determinants/matchings and exact Sturm gap enclosures.
graph_results={}
polys={
 'chr':[1,-21,180,-824,2214,-3605,3533,-1990,577,-64],
 'tet':[1,-21,180,-822,2192,-3510,3321,-1731,415,-25]}
intervals={'chr':(Q(10402,10000),Q(10403,10000)),
           'tet':(Q(5899,10000),Q(5900,10000))}
for name,a,k in [('chr',A_chr,8),('tet',A_tet,5)]:
    check(len(a)==18 and sum(map(sum,a))==42,'graph')
    for i,j in product(range(18),repeat=2):
        check(a[i][j]==a[j][i],'graph')
        if (i<9)==(j<9):check(a[i][j]==0,'bipartite')
    b=[row[9:] for row in a[:9]]
    match=sum(all(b[i][p[i]] for i in range(9)) for p in permutations(range(9)))
    check(match==k,'matching');det=bareiss(a)
    check(det==-k*k,'determinant');check(bareiss(b)==k,'determinant')
    poly=charpoly(matmul(b,list(map(list,zip(*b)))))
    check(poly==polys[name],'charpoly')
    seq=sturm(poly);lo,hi=intervals[name]
    check(rootcount(seq,Q(0),(lo/2)**2)==0,'sturm_gap')
    check(rootcount(seq,(lo/2)**2,(hi/2)**2)==1,'sturm_gap')
    ev=jacobi_evals(a);gap=ev[9]-ev[8]
    check(float(lo)<gap<float(hi),'numerical_gap',True)
    check(max(abs(ev[i]+ev[-1-i]) for i in range(18))<1e-10,'numerical_gap',True)
    graph_results[name]={'matchings':match,'determinant':det,'gap':gap,
     'gap_interval':[str(lo),str(hi)],'squared_spectrum_polynomial':poly}
ratio=graph_results['chr']['gap']/graph_results['tet']['gap']
check(round(ratio,4)==1.7634,'numerical_gap',True)
check(abs(graph_results['tet']['gap']-(math.sqrt(7-2*math.sqrt(5))-1))<1e-11,'
    numerical_gap',True)
# V4: risk surrogate, finite entropy, static diagonal norm and budget.
def risk(x):return 2/(1+math.exp(x))
def h2(p):return -p*math.log2(p)-(1-p)*math.log2(1-p)
gates={}
for p in [.95,.99]:
    x=-math.log(p**(-.2)-1);q=1/(1+math.exp(x));a=2*q
    check(abs((1-q)**5-p)<1e-12,'gate',True)
    check(0<q<.5 and 5*h2(q)>0,'finite_bias_entropy',True)
    gates[str(p)]={'dimensionless_bias':x,'risk':a,'attenuation':1/a,
                   'entropy_bits':5*h2(q),'meV_at_300K':x*8.617333262145e-2*300}
for m in range(1,5):
    for u in product(range(5),repeat=m):
        check(min(u)*m<=sum(u),'budget_order')
        check(max(map(risk,u))+1e-14>=risk(sum(u)/m),'budget_risk',True)
```

```
        weights=[Q(2,1+x) for x in [z+1 for z in u]]
        norm=max(weights)
        for w in product([-1,0,1],repeat=m):
            check(sum((a*b)**2 for a,b in zip(weights,w))<=norm**2*sum(z*z for z in w),
                'diagonal_norm')
        j=weights.index(norm);check(weights[j]**2==norm**2,'norm_attainment')
for x in product(range(4),repeat=4):
    check((x[0]^x[1])^(x[1]^x[2])^(x[2]^x[3])^(x[3]^x[0])==0,'plaquette')
check(2*2*6+2*3*5+3*2*5==84,'plaquette_count')
# V5: actual support-poset categories, target word0, two species.
# Hom(Y1,Y2) iff same species and support2 subset support1.
supports=[s for s in range(16) if s&1]
objects=list(product(range(2),supports))
def subset(a,b):return a&b==a
def hom(y,z):return y[0]==z[0] and subset(z[1],y[1])
def F(s):return (s,15)
def G(y):return y[0]
for y in objects:
    check(hom(y,y),'category_identity');check(hom(F(G(y)),y),'counit')
    for x in range(2):check(hom(F(x),y)==(x==G(y)),'adjunction_hom')
    for z in objects:
        if hom(y,z):check(G(y)==G(z),'functor_G')
        for w in objects:
            if hom(y,z) and hom(z,w):check(hom(y,w),'category_composition')
for x in range(2):
    check(G(F(x))==x,'unit');check(hom(F(G(F(x))),F(x)),'triangle_F')
for y in objects:check(G(F(G(y)))==G(y),'triangle_G')
check(3**6==729 and 2**5==32,'category_object_counts')
# V6: indicator feasibility, not unique geometry/Euler-Lagrange dynamics.
for a,b,c in product([False,True],repeat=3):
    check((a and b and c)==(sum([a,b,c])==3),'feasibility')
# V7: boundary capacity under stipulated alphabet/packing law.
# sqrt and pi enclosed rationally, so the tile floor is certified.
L=233;nlat=54;c=Q(89923,100000);d=4;area=Q(1,2)
scale=10**12;s=math.isqrt(L*scale*scale)
slo,shi=Q(s,scale),Q(s+1,scale)
pilo,pihi=Q(3141592653589793238,10**18),Q(3141592653589793239,10**18)
rlo,rhi=c*slo-d,c*shi-d
alo,ahi=4*pilo*rlo*rlo,4*pihi*rhi*rhi
flo,fhi=(alo/area).__floor__(),(ahi/area).__floor__()
check(flo==fhi,'capacity_floor');hint=2*L+4*nlat;capacity=2*flo
check(hint<capacity,'capacity')
# All-L sufficient bound for L>=233: R>=.63 sqrt(L), pi>3.
check((c-Q(63,100))**2*233>16,'capacity_all_L')
coef=24*Q(63,100)**2
check(coef>1 and 2*(coef-1)*233>218,'capacity_all_L')
capacity_result={'L':L,'nodes':nlat,'internal_bits':hint,'tiles':flo,
 'capacity_bits':capacity,'utilization_percent':100*hint/capacity,
 'radius_nm':float((rlo+rhi)/2),'asymptotic_percent':100*float(area)/(4*math.pi*float(c
    )**2)}
# Ten intentionally wrong alternatives must be rejected.
negative(round(ratio,4)==1.7633)
negative(bareiss(A_chr)==-25)
negative((4&3)==fmul(2,2,7,2))
negative(3**6==32)
negative(hom((0,1),(0,15)))
negative(hom((0,15),(1,1)))
negative(5*h2(1/(1+math.exp(5)))==0)
negative((True and True and False)==True)
negative(hint==660)
negative(flo==2220)
report={'article':97,'checks_total':COUNTS['exact']+COUNTS['numerical'],
 'exact_checks':COUNTS['exact'],'numerical_checks':COUNTS['numerical'],
 'negative_controls_included':COUNTS['negative_controls'],'by_block':dict(COUNTS),
 'graphs':graph_results,'gap_ratio':ratio,'gates':gates,'capacity':capacity_result,
 'scope':'Finite arithmetic and model checks; graph spectral enclosures exact. No
    chemistry, general control synthesis, gravitational entropy or variational
    uniqueness is certified.'}
```

```
Path('verification_report.json').write_text(json.dumps(report,indent=2,sort_keys=True)+'\n')
print(json.dumps(report,indent=2,sort_keys=True))

macros={'valSlackZeroMicro':1,'valSlackZeroEnt':0,'valSlackRegSix':31,
 'valMicroRegSix':32,'valEntRegSix':5,'valStrainThreeDThree':162,
 'valStrainTwoDThree':36,'valStrainSavedThree':126,'valKekuleChr':8,
 'valDetChr':-64,'valGapChr':f"{graph_results['chr']['gap']:.4f}",
 'valKekuleTet':5,'valDetTet':-25,'valGapTet':f"{graph_results['tet']['gap']:.4f}",
 'valGapRatio':f"{ratio:.4f}",'valHinfEndo':'1.0000','valGammaInfEndo':'0.5000',
 'valHinfPlaceNinetyFive':f"{gates['0.95']['risk']:.4f}",
 'valAttenNinetyFive':f"{gates['0.95']['attenuation']:.1f}",
 'valHinfPlaceNinetyNine':f"{gates['0.99']['risk']:.4f}",
 'valAttenNinetyNine':f"{gates['0.99']['attenuation']:.1f}",
 'valHomCard':1,'valCZeroObjectsSix':729,'valLcritBp':233,'valLatNodes':54,
 'valHintBits':hint,'valNpartialTiles':f'{flo:,}','valHholoBits':f'{capacity:,}',
 'valHoloRatioPct':f"{capacity_result['utilization_percent']:.2f}",
 'valHoloAsympPct':f"{capacity_result['asymptotic_percent']:.2f}"}
Path('verification_values_a97.tex').write_text('% Generated by revised verify_a97.py\n'+
 ''.join('\\newcommand{\\'+k+'}{'+str(v)+'}\n' for k,v in macros.items()))
```