Formal Proof Dossier

Article 99

Endogenous and Exogenous Sources of Encoded Chrysene Systems

Functional Selection, Adaptive Realization, and Engineered Emulation

Charles D. Schaper, Ph.D.

Volume 1, Number 4 November 2026

Certification Summary

The original scientific formalization is retained with the Article99 namespace. Fresh Lean 4.24 compilation and audits pass for 25 named theorems: nine are axiom-free and sixteen use only standard foundational axioms. This renumbering does not extend the scope of the proofs.

Publication integration

This companion belongs to the revised manuscript formerly numbered Article 102 and now designated Article 99. The renumbering and presentation update preserve the supplied scientific content. Execution evidence below was refreshed for the renamed source. The historical package remains the provenance source; this is not a new independent peer review.

1 Status and reproducibility

The accompanying `FormalProof_A99.lean` was compiled with the official Lean 4.24.0 Linux release with exit status zero. The exact release commit is:

```
797c613eb9b6d4ec95db23e3e00af9ac6657f24b
```

The source imports only `Std` and `Init.Data.Rat.Lemmas`; Mathlib is not required. In a folder containing the source, run:

```
lean --version
lean -o FormalProof_A99.olean FormalProof_A99.lean
```

The bundled `audit_lean.py` runs these commands and also accepts an explicit executable through the `LEAN` environment variable. A successful LaTeX compilation does not run Lean. The included `lean_audit.log` records the separate Lean run.

There are no unproved declarations, no `sorry`, no custom axioms, and no use of `native_decide`. Finite enumeration uses `decide +kernel`, so its proof certificates are checked by the kernel rather than accepted on the authority of native execution. Every theorem has a printed axiom audit. The only dependencies reported are the standard Lean foundational axioms `propext`, `Classical.choice`, and `Quot.sound`; several structural and enumeration results require none. In particular, no audit contains `sorryAx` or an external computation axiom.

Source integrity (SHA-256).

```
8d3a760f1f78eef9ec2de31583fb0f0b2a296fc11c2504280fd8b4b40cd90436
```

2 Correspondence and precise scope

Manuscript subject	Machine-checked scope
Reachability and emulation	Generic image equality, objective transfer, transitivity.
Selection and acceptance	Generic rational normalization; exact hexamer accepted law.
Doob transforms	Generic rational row normalization and all finite-horizon telescoping.
Unknown-demand bound	Integer-weight averaging obstruction only.
Hexamer registry	Exhaustive exact six-bit fibers and five-bit reversal census.
KL, spectral and continuous-time analysis	Not formalized; manuscript proofs remain separate.

1. **Common observations and emulation.** `observation_reachability` proves equality of observed images under a surjective placement map. `emulation_transfer` preserves the order $\forall u\, \exists v\, \forall e$ and proves equality after every common environment-dependent objective. `emulation_trans` proves transitivity. The types of protocols must already encode admissibility and any required resource restrictions. No causal or chemical feasibility is inferred merely from these type-level statements.

2. **Functional-fiber algebra.** `mass_scale`, `conditional_normalizes`, `lifted_fiber_mass`, and `lift_total_mass` prove, for arbitrary finite lists and rational weights, the normalization identities underlying finite disintegration and the lifted law. The list model counts multiplicities: an interpretation as a finite configuration set therefore requires duplicate-free exhaustive enumerations and disjoint fibers. The generic lemmas prove algebraic normalization, not nonnegativity or the partition property. For probability laws these are separate premises. They do not prove the relative-entropy minimization theorem.

3. **Acceptance.** `accepted_mass` and `accepted_conditional` prove scaling and cancellation for a normalized rational law. Together with the concrete acceptance certificates they verify the exact accepted distribution in the hexamer example. They do not prove the general rejection mechanism's optimality, geometric-tail law, or almost-sure termination.

4. **Backward transforms.** `doob_row_normalizes` derives unit row sum from the backward recursion and a nonzero normalizer. `doob_path_telescopes` and `doob_conditioned_path` prove the pointwise path identity at every finite horizon, for arbitrary rational edge weights and a nonzero h along the path. These are generic proofs, not sample-path tests. They do not certify positivity, inaccessible zero-h rows, path-law entropy optimality, or existence of the backward solution. Zero-h boundary cases remain in the manuscript's mathematical proof.

5. **Restricted minimax arithmetic.** `minimax_upper_integer` proves that a nonempty list of nonnegative integer weights with total at most mk contains a weight at most k. This is the integer-weight threshold form of the averaging obstruction, not a formalization of the full real-valued minimax theorem or its uniqueness statement.

6. **Exact registry census and balanced selection.** The six-bit map has two preimages for every five-bit agreement word; complementing all six bits preserves the agreement word. Reversal of the 32 words is an involution. There are eight fixed words, twenty canonical representatives and twelve nontrivial pairs. The rational law assigning 1/20 to a palindrome and 1/40 to every other word is normalized, with mass 1/20 per reversal class. Acceptance probabilities 1 and 1/2 give overall acceptance 5/8, recover that law exactly, and have reciprocal 8/5. The final reciprocal is an arithmetic identity, not by itself a proof that an expected stopping time equals the reciprocal.

3 Claims not machine checked here

The logarithmic chain rule, Gibbs inequality, uniqueness of the information projection, entropy maximization, general acceptance optimum, total-variation error bounds, robust-value transfer, continuous-time response, spectral-gap comparison and mixing estimate, general graph cohomology/rank formulas, quantum-channel statements, and chemical realizability are not certified by this Lean module. Their manuscript proofs and computational checks have distinct evidentiary status. In particular, no analytic theorem has been replaced by an axiom asserting its conclusion. The formalization neither proves biogenesis nor establishes that a physical endogenous or engineered source meets the declared hypotheses.

4 Complete Lean source

FormalProof_A99.lean

```
import Std
import Init.Data.Rat.Lemmas

/-!
Article 99: kernel-checked structural and exact rational results.
Lean 4.24.0; no Mathlib, no sorry, no added axioms, no native_decide.
Analytic relative entropy, spectral gaps and CTMC convergence are NOT
formalized by this module. Lists enumerate finite spaces; normalization
lemmas are valid even with multiplicities (use duplicate-free lists for sets).
-/
namespace Article99

-- Correct quantifier order: a single emulator works in every environment.
def Emulates {U V E L : Type} (p : U → E → L) (q : V → E → L) : Prop :=
  ∀ u, ∃ v, ∀ e, p u e = q v e

theorem emulation_transfer {U V E L R : Type}
    {p : U → E → L} {q : V → E → L}
    (h : Emulates p q) (objective : E → L → R) :
    ∀ u, ∃ v, ∀ e, objective e (p u e) = objective e (q v e) := by
  intro u
  obtain ⟨v, hv⟩ := h u
  exact ⟨v, fun e => congrArg (objective e) (hv e)⟩

theorem emulation_trans {U V W E L : Type}
    {p : U → E → L} {q : V → E → L} {r : W → E → L}
    (hpq : Emulates p q) (hqr : Emulates q r) : Emulates p r := by
  intro u
  obtain ⟨v, hv⟩ := hpq u
  obtain ⟨w, hw⟩ := hqr v
  exact ⟨w, fun e => (hv e).trans (hw e)⟩

theorem observation_reachability {U X Y : Type} (placement : U → X)
    (onto : ∀ x, ∃ u, placement u = x) (observe : X → Y) (y : Y) :
    (∃ x, observe x = y) ↔ (∃ u, observe (placement u) = y) := by
  constructor
  · rintro ⟨x, hx⟩
    obtain ⟨u, hu⟩ := onto x
    exact ⟨u, by rw [hu]; exact hx⟩
  · rintro ⟨u, hu⟩
```

```
    exact ⟨placement u, hu⟩

-- A finite sum, kept explicit to make the represented measure unambiguous.
def mass {X : Type} (xs : List X) (f : X → Rat) : Rat :=
  (xs.map f).foldr (· + ·) 0

theorem mass_scale {X : Type} (xs : List X) (f : X → Rat) (a : Rat) :
    mass xs (fun x => a * f x) = a * mass xs f := by
  induction xs with
  | nil => simp [mass, Rat.mul_zero]
  | cons x xs ih =>
    change a * f x + mass xs (fun x => a * f x) = a * (f x + mass xs f)
    rw [ih, Rat.mul_add]

theorem conditional_normalizes {X : Type} (xs : List X) (mu : X → Rat)
    (h : mass xs mu ≠ 0) :
    mass xs (fun x => (1 / mass xs mu) * mu x) = 1 := by
  rw [mass_scale]
  exact Rat.div_mul_cancel h

theorem lifted_fiber_mass {X : Type} (xs : List X) (mu : X → Rat)
    (r : Rat) (h : mass xs mu ≠ 0) :
    mass xs (fun x => (r / mass xs mu) * mu x) = r := by
  rw [mass_scale]
  exact Rat.div_mul_cancel h

-- A partition is represented as a dependent family of nonempty fiber lists.
-- The following theorem proves full normalization after fiber disintegration.
theorem lift_total_mass {X Y : Type} (ys : List Y) (fiber : Y → List X)
    (mu : X → Rat) (r : Y → Rat)
    (positive : ∀ y, mass (fiber y) mu ≠ 0) (hr : mass ys r = 1) :
    mass ys (fun y => mass (fiber y)
      (fun x => (r y / mass (fiber y) mu) * mu x)) = 1 := by
  have heq : (fun y => mass (fiber y)
      (fun x => (r y / mass (fiber y) mu) * mu x)) = r := by
    funext y
    exact lifted_fiber_mass (fiber y) mu (r y) (positive y)
  rw [heq, hr]

-- If acceptance weights are c times a normalized law, conditioning recovers
-- that law. This proves the algebraic step, not almost-sure termination.
theorem accepted_mass {X : Type} (xs : List X) (p : X → Rat)
    (hp : mass xs p = 1) (c : Rat) :
    mass xs (fun x => c * p x) = c := by
  rw [mass_scale, hp, Rat.mul_one]

theorem accepted_conditional {X : Type} (p : X → Rat) (c : Rat)
    (hc : c ≠ 0) (x : X) : (p x * c) / c = p x :=
  Rat.mul_div_cancel hc

-- Backward-h row normalization, for arbitrary rational transition weights.
-- Positivity and probabilistic interpretation remain explicit external premises.
theorem doob_row_normalizes {X : Type} (xs : List X)
    (kernel nextH : X → Rat) (h : Rat)
    (backward : mass xs (fun z => kernel z * nextH z) = h)
    (nonzero : h ≠ 0) :
    mass xs (fun z => (1/h) * (kernel z * nextH z)) = 1 := by
  rw [mass_scale, backward]
  exact Rat.div_mul_cancel nonzero
```

```
def pathProduct (k : Nat → Rat) : Nat → Rat
  | 0 => 1
  | n + 1 => pathProduct k n * k n

def tilted (k h : Nat → Rat) (n : Nat) : Rat :=
  k n * h (n+1) / h n

-- Arbitrary finite horizon, not a numerical example. The assumption describes
-- a positive-h path; it does not assert the telescoping conclusion.
theorem doob_path_telescopes (k h : Nat → Rat) (n : Nat)
    (nonzero : ∀ t, t ≤ n → h t ≠ 0) :
    h 0 * pathProduct (tilted k h) n = pathProduct k n * h n := by
  induction n with
  | zero => simp [pathProduct, Rat.mul_one, Rat.one_mul]
  | succ n ih =>
    have hn : h n ≠ 0 := nonzero n (by omega)
    have hi := ih (fun t ht => nonzero t (by omega))
    simp only [pathProduct, tilted]
    grind

theorem doob_conditioned_path (k h : Nat → Rat) (n : Nat)
    (nonzero : ∀ t, t ≤ n → h t ≠ 0) (nu a : Rat) :
    (nu * h 0 / a) * pathProduct (tilted k h) n =
      (nu * pathProduct k n * h n) / a := by
  have ht := doob_path_telescopes k h n nonzero
  grind

-- Integer-weight form of the finite unknown-demand minimax upper bound.
-- Dividing by total mass yields min probability <= 1 / number of demands.
theorem mass_above_threshold (ws : List Nat) (k : Nat)
    (h : ∀ w ∈ ws, k < w) : ws.length * (k+1) ≤ ws.sum := by
  induction ws with
  | nil => simp
  | cons w ws ih =>
    have hw := h w (by simp)
    have hi := ih (fun v hv => h v (by simp [hv]))
    simp only [List.length_cons, List.sum_cons, Nat.add_mul]
    omega

theorem minimax_upper_integer (ws : List Nat) (k : Nat)
    (hne : ws ≠ []) (htotal : ws.sum ≤ ws.length * k) :
    ∃ w ∈ ws, w ≤ k := by
  apply Classical.byContradiction
  intro hn
  have hall : ∀ w ∈ ws, k < w := by
    intro w hw
    have hh : ¬ w ≤ k := by intro hk; exact hn ⟨w, hw, hk⟩
    omega
  have hbound := mass_above_threshold ws k hall
  have hlen : 0 < ws.length := List.length_pos_iff.mpr hne
  simp only [Nat.mul_add, Nat.mul_one] at hbound
  omega

-- Five agreement bits of a binary hexamer modulo global flip.
def edgeBit (n i : Nat) : Nat :=
  ((n / 2^i) % 2 + (n / 2^(i+1)) % 2) % 2
def agreement5 (n : Nat) : Nat :=
  edgeBit n 0 + 2 * edgeBit n 1 + 4 * edgeBit n 2 +
```

```
  8 * edgeBit n 3 + 16 * edgeBit n 4

theorem binary_hexamer_fiber_size : ∀ a ∈ List.range 32,
    ((List.range 64).filter (fun n => agreement5 n == a)).length = 2 := by decide +kernel

theorem global_flip_preserves_agreements : ∀ n ∈ List.range 64,
    agreement5 (63-n) = agreement5 n := by decide +kernel

def reverse5 (n : Nat) : Nat :=
  ((n / 1) % 2) * 16 + ((n / 2) % 2) * 8 +
  ((n / 4) % 2) * 4 + ((n / 8) % 2) * 2 + ((n / 16) % 2)
def words : List Nat := List.range 32
def palindromes : List Nat := words.filter (fun n => n == reverse5 n)
def representatives : List Nat := words.filter (fun n => n ≤ reverse5 n)
def balancedWeight (n : Nat) : Rat := if n == reverse5 n then 1/20 else 1/40
def acceptance (n : Nat) : Rat := if n == reverse5 n then 1 else 1/2

theorem reversal_is_involution :
    ∀ n ∈ words, reverse5 (reverse5 n) = n := by decide +kernel

theorem palindrome_count : palindromes.length = 8 := by decide +kernel
theorem orbit_count : representatives.length = 20 := by decide +kernel
theorem paired_orbit_count :
    (representatives.filter (fun n => n != reverse5 n)).length = 12 := by decide +kernel

theorem balanced_normalization : mass words balancedWeight = 1 := by decide +kernel
theorem each_class_balanced : ∀ n ∈ representatives,
    (if n == reverse5 n then balancedWeight n
      else balancedWeight n + balancedWeight (reverse5 n)) = 1/20 := by decide +kernel

theorem total_acceptance :
    mass words (fun n => (1/32 : Rat) * acceptance n) = 5/8 := by decide +kernel

theorem accepted_law_exact : ∀ n ∈ words,
    ((1/32 : Rat) * acceptance n) / (5/8) = balancedWeight n := by decide +kernel

theorem reciprocal_acceptance : (1 : Rat) / (5/8) = 8/5 := by decide +kernel

end Article99

-- Full axiom audit of every theorem in this module.
#print axioms Article99.emulation_transfer
#print axioms Article99.emulation_trans
#print axioms Article99.observation_reachability
#print axioms Article99.mass_scale
#print axioms Article99.conditional_normalizes
#print axioms Article99.lifted_fiber_mass
#print axioms Article99.lift_total_mass
#print axioms Article99.accepted_mass
#print axioms Article99.accepted_conditional
#print axioms Article99.doob_row_normalizes
#print axioms Article99.doob_path_telescopes
#print axioms Article99.doob_conditioned_path
#print axioms Article99.mass_above_threshold
#print axioms Article99.minimax_upper_integer
#print axioms Article99.binary_hexamer_fiber_size
#print axioms Article99.global_flip_preserves_agreements
#print axioms Article99.reversal_is_involution
#print axioms Article99.palindrome_count
```

```
#print axioms Article99.orbit_count
#print axioms Article99.paired_orbit_count
#print axioms Article99.balanced_normalization
#print axioms Article99.each_class_balanced
#print axioms Article99.total_acceptance
#print axioms Article99.accepted_law_exact
#print axioms Article99.reciprocal_acceptance
```