Formal Proof Dossier

Article 104: From Registry-Controlled Capture to Productive Molecular Assembly

Constructive Existence of a Minimal Abiotic Loom

Annals of the Chrysene Formalism, Vol. 1, No. 5
November 2026

Kernel-checked partial formalization

Proof source	`Article104.lean`
Tested toolchain	Lean 4.24.0
Imports	`Std`, `Init.Data.Rat.Lemmas`
Named theorems	**49**
Theorem dependency audits	**49 passed**
Custom axioms / proof holes	0 / 0
`native_decide` / unsafe declarations	0 / 0
Compiler exit status	0

What this dossier certifies

This is a **partial formalization** of the finite structural and rational core of Article 104. It is not a kernel proof of the entire manuscript or of a chemically realized loom. Some theorems use no axioms; others use only the standard Lean foundations `propext`, `Classical.choice`, and `Quot.sound`. "No custom axioms" must not be read as "no foundational axioms."

Reproduction and provenance

The executed compiler was Lean 4.24.0, commit `797c613eb9b6d4ec95db23e3e00af9ac6657f24b`. No Mathlib installation is required. With that toolchain active, save the complete UTF-8 source below as `Article104.lean` and run:

```
lean Article104.lean
python3 audit.py /path/to/lean
```

The second command, available in the source package, recompiles the source, reads every printed dependency audit, and regenerates `audit_summary.json`. Kernel computation uses `decide +kernel`; no native-decision shortcut, custom axiom, proof hole, or unsafe override is present. The supplied source was recompiled and its dependency audit repeated for this publication package.

Overleaf compilation typesets this document only. It does not run Lean. The certification above refers to actual separate kernel executions. For execution use the supplied raw UTF-8 file rather than relying on text extracted from a PDF.

Source SHA-256:
`e3863a4dbf60391a9ae5f7d7399896083001b4605ceb4a22fb41b1f65d7040d7`

Theorem index by coverage group

Group	Kernel-checked content	Boundary of the formal result
F1: execution	Generic finite execution composition, invariant propagation, protected projection, origin-label preservation	Does not prove that a proposed physical transition satisfies these interfaces.
F2: joining	Per-atom integer incidence balance for the two deleted and two inserted bonds; unchanged degree	Bond existence/absence, endpoint distinctness, element valence, and full graph chemistry remain separate guards.
F3: coding	Encoder/decoder identity for arbitrary G/A words, actual-symbol append, injectivity, and freshness by unused position	Abstract external symbols; no molecular writer or nucleobase insertion.
F4: formulas	Positive-length elemental and total-atom balances, all 16 tetramer cases, target/alternate formulas, 92-atom common pool	Formula balance is weaker than the complete explicit atom-origin graph certificate in Python.
F5: geometry	Intact node/edge counts, all pairwise core-center separations, skew basis, packet gap, signatures	No full continuous transport or tether-collision proof.
F6: kinetics	Generic three-stage probability partition identities, exact nominal/robust bounds, all 512 corners, mean-time arithmetic	No CTMC existence, exponential waiting, or continuous-box monotonicity proof is formalized here.
F7: receiver	Rational square completion, nonnegative/positive squares, global minimum, attainment, uniqueness, box feasibility, energies, rational normalization/balance	Rational coordinates only; real analysis, positivity-based mixing, and molecular interactions remain paper-level claims.
F8: Taylor arithmetic	Exact degree-40 rational sums/tails, rational enclosure expressions, final contrast and readout-error inequalities	The bridge $S_N < \exp \leq U_N$ is analytic and not formalized. No real-exponential theorem is silently inferred from rational computation.

Relationship to the paper's principal claims

The paper proves its real-variable mechanical and finite-state stochastic statements analytically. The Lean file verifies selected exact algebraic, finite combinatorial, and structural statements used in those proofs. In particular:

- The formal join-degree theorem balances endpoint incidences. It does not certify the whole atom-labelled joining operation without the specified graph guards.
- The corner theorem checks 512 finite cases. The proof over the entire independent rate box is the paper's monotonicity argument.
- The rational receiver minimum is a theorem over `Rat`. Its use within a real mechanical model additionally relies on the paper's real square-completion argument.

- The three "rational enclosure" theorems compare rational Taylor-certificate expressions. The analytic Taylor remainder and reciprocal monotonicity supply their connection to physical or real-exponential occupancies.
- The newly explicit phase refinement is covered computationally by the Python protocol tests and structurally by generic execution invariants, not by a claimed full Lean formalization of that implementation.

Neither formal correctness nor numerical verification supplies amide activation chemistry, a chrysene-stock preparation map, empirical electrostatics, a physical encoding apparatus, or autonomous operation.

Complete Lean 4 source

The source includes all definitions, all 49 theorem declarations, and one dependency-audit command for each theorem. Unicode symbols are typeset for readability; the raw source is supplied unchanged in the package.

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

/-!
Article 104: kernel-checked PARTIAL formalization of the declared reduced model.
Lean 4.24.0 / Std only. Rational numbers below are dimensionless.
This file proves finite, ledger, coding, and rational algebra statements. It does
not prove chemical feasibility, real-exponential bounds, CTMC convergence,
continuous routing, or a physical atomistic synthesis trajectory.
-/
namespace Article104
set_option maxRecDepth 100000
set_option maxHeartbeats 0

inductive Execution {X : Type} (step : X → X → Prop) : X → X → Prop
  | empty (x) : Execution step x x
  | next {x y z} : step x y → Execution step y z → Execution step x z

theorem execution_composition {X : Type} {step : X → X → Prop} {x y z : X}
    (hxy : Execution step x y) (hyz : Execution step y z) : Execution step x z := by
  induction hxy with
  | empty => exact hyz
  | next h _ ih => exact .next h (ih hyz)

theorem execution_preserves_invariant {X : Type} {step : X → X → Prop}
    (I : X → Prop) (preserves : ∀ x y, step x y → I x → I y)
    {x y : X} (h : Execution step x y) (hx : I x) : I y := by
  induction h with
  | empty => exact hx
  | next h _ ih => exact ih (preserves _ _ h hx)

theorem protected_composition {X P : Type} (projection : X → P)
    {x y z : X} (hxy : projection x = projection y) (hyz : projection y = projection z) :
    projection x = projection z := hxy.trans hyz

theorem protected_execution {X P : Type} (projection : X → P)
    (step : X → X → Prop) (fixed : ∀ x y, step x y → projection x = projection y)
    {x y : X} (h : Execution step x y) : projection x = projection y := by
  induction h with
  | empty => rfl
  | next h _ ih => exact (fixed _ _ h).trans ih

-- Ledger origin labels persist when only connectivity changes.
structure AtomLedger (Atom Packet : Type) where
  origin : Atom → Packet
  bond : Atom → Atom → Nat
```

```
47
48 def changeBonds {A P : Type} (l : AtomLedger A P) (b : A → A → Nat) : AtomLedger A P :=
49   { l with bond := b }
50
51 theorem origin_preserved {A P : Type} (l : AtomLedger A P) (b : A → A → Nat) (a : A) :
52     (changeBonds l b).origin a = l.origin a := rfl
53
54 -- Integer incidence balance of the two-deletion/two-addition join.
55 -- C and N each lose/gain one bond; water O and transferred H do likewise.
56 def degreeUpdate (degree removed added : Int) : Int := degree - removed + added
57
58 theorem balanced_join_degree (degree : Int) : degreeUpdate degree 1 1 = degree := by
59   unfold degreeUpdate; omega
60
61 theorem unaffected_join_degree (degree : Int) : degreeUpdate degree 0 0 = degree := by
62   unfold degreeUpdate; omega
63
64 theorem four_endpoint_balance (c n o h : Int) :
65     (degreeUpdate c 1 1, degreeUpdate n 1 1, degreeUpdate o 1 1, degreeUpdate h 1 1)
66       = (c,n,o,h) := by simp [balanced_join_degree]
67
68 -- Incidence of an undirected single bond. The join's removed and inserted
69 -- edge multisets have identical incidence at EVERY atom. Application to a
70 -- valid simple bond graph additionally requires the paper's edge guards.
71 def incidence {A : Type} [DecidableEq A] (v : A) (e : A × A) : Int :=
72   (if v = e.1 then 1 else 0) + (if v = e.2 then 1 else 0)
73
74 theorem join_incidence_balance {A : Type} [DecidableEq A] (v c o n h : A) :
75     incidence v (c,o) + incidence v (n,h) =
76       incidence v (c,n) + incidence v (o,h) := by
77   simp only [incidence]
78   omega
79
80 theorem guarded_join_preserves_degree {A : Type} [DecidableEq A]
81     (v c o n h : A) (degree : Int) :
82     degree - incidence v (c,o) - incidence v (n,h) +
83       incidence v (c,n) + incidence v (o,h) = degree := by
84   have hh := join_incidence_balance v c o n h
85   omega
86
87 inductive Residue | G | A deriving DecidableEq, Repr
88 inductive Symbol | CG | CA deriving DecidableEq, Repr
89 open Residue Symbol
90
91 def encodeResidue : Residue → Symbol | G => CG | A => CA
92 def decodeSymbol : Symbol → Residue | CG => G | CA => A
93 def encode (w : List Residue) := w.map encodeResidue
94 def decode (c : List Symbol) := c.map decodeSymbol
95
96 theorem decode_encode (w : List Residue) : decode (encode w) = w := by
97   induction w with
98   | nil => rfl
99   | cons r w ih => cases r <;> simp_all [encode, decode, encodeResidue, decodeSymbol]
100
101 theorem actual_prefix_append (c : List Symbol) (w : List Residue) (r : Residue)
102     (h : decode c = w) : decode (c ++ [encodeResidue r]) = w ++ [r] := by
103   cases r <;> simp_all [decode, encodeResidue, decodeSymbol]
104
105 theorem code_injective (u v : List Residue) (h : encode u = encode v) : u = v := by
106   have hh := congrArg decode h
107   simpa [decode_encode] using hh
108
109 -- Resource identity is position plus species. Choosing a fresh position cannot
110 -- reuse a packet, regardless of the species selected at that position.
111 abbrev Packet := Nat × Residue
112
113 theorem packet_fresh (j : Nat) (r : Residue) (used : List Packet)
114     (h : ∀ p ∈ used, p.1 < j) : (j,r) ∉ used := by
115   intro hm
```

```
  have hh := h (j,r) hm
  omega

def words : Nat → List (List Residue)
  | 0 => [[]]
  | n+1 => (words n).flatMap fun w => [G :: w, A :: w]
def alanines (w : List Residue) : Nat := (w.filter (· == A)).length
-- Formula ordering is C,H,N,O.
def productFormula (n a : Nat) := (2*n+a, 3*n+2*a+2, n, n+1)
def inputFormula (n a : Nat) := (2*n+a, 5*n+2*a, n, 2*n)
def atomCount (f : Nat × Nat × Nat × Nat) := f.1+f.2.1+f.2.2.1+f.2.2.2

theorem peptide_formula_balance (n a : Nat) (hn : 1 ≤ n) :
    let p := productFormula n a
    let f := inputFormula n a
    p.1 = f.1 ∧ p.2.1+2*(n-1) = f.2.1 ∧ p.2.2.1 = f.2.2.1 ∧
      p.2.2.2+(n-1) = f.2.2.2 := by
  simp [productFormula, inputFormula]; omega

theorem peptide_atom_balance (n a : Nat) (hn : 1 ≤ n) :
    atomCount (productFormula n a)+3*(n-1) = atomCount (inputFormula n a) := by
  simp [atomCount, productFormula, inputFormula]; omega

theorem sixteen_words : (words 4).length = 16 ∧ (words 4).Nodup := by decide +kernel

theorem all_tetramers_conserve : ∀ w ∈ words 4,
    atomCount (productFormula w.length (alanines w))+9 =
      atomCount (inputFormula w.length (alanines w)) := by decide +kernel

theorem target_formula : productFormula 4 (alanines [G,A,G,A]) = (10,18,4,5) ∧
    atomCount (productFormula 4 2) = 37 ∧ atomCount (inputFormula 4 2) = 46 := by decide +kernel

theorem alternate_formula : productFormula 4 (alanines [G,G,G,A]) = (9,16,4,5) ∧
    atomCount (productFormula 4 1) = 34 := by decide +kernel

def pool : List Packet := (List.range 4).flatMap fun j => [(j,G),(j,A)]
def packetSize (p : Packet) : Nat := match p.2 with | G => 10 | A => 13

theorem common_pool : pool.length = 8 ∧ pool.Nodup ∧
    (pool.map packetSize).sum = 92 := by decide +kernel

def signature : List Residue → Nat
  | [] => 0
  | r::w => (if r = A then 1 else 0) + 2*signature w

theorem signature_catalogue : ((words 4).map signature).Nodup ∧
    ∀ w ∈ words 4, signature w ≤ 15 := by decide +kernel

theorem signatures_distinguish : signature [G,A,G,A] = 10 ∧
    signature [G,G,G,A] = 8 := by decide +kernel

-- Exact full-patch geometry. This checks centers and adjacency, not tether paths.
abbrev Node := Fin 4 × Fin 2 × Fin 2
abbrev Vec := Int × Int × Int

def center (p : Node) : Vec := (4*(p.1.val : Int)+(p.2.1.val : Int),
  4*(p.2.1.val : Int),4*(p.2.2.val : Int))
def nodes : List Node := (List.finRange 4).flatMap fun i =>
  (List.finRange 2).flatMap fun j => (List.finRange 2).map fun k => (i,j,k)
def lateral (p q : Node) : Bool := p.2.2 == q.2.2 &&
  ((p.1.val+1 == q.1.val && p.2.1 == q.2.1) ||
   (p.2.1.val+1 == q.2.1.val && p.1 == q.1))
def stacked (p q : Node) : Bool := p.1 == q.1 && p.2.1 == q.2.1 &&
  p.2.2.val+1 == q.2.2.val
def edgeCount (edge : Node → Node → Bool) :=
  (nodes.flatMap fun p => nodes.filter fun q => edge p q).length
def sqDistance (p q : Vec) := (p.1-q.1)*(p.1-q.1)+
  (p.2.1-q.2.1)*(p.2.1-q.2.1)+(p.2.2-q.2.2)*(p.2.2-q.2.2)

```

```
theorem full_patch_count : nodes.length = 16 ∧ nodes.Nodup := by decide +kernel
theorem full_patch_edges : edgeCount lateral = 20 ∧ edgeCount stacked = 8 := by decide +kernel
theorem full_patch_separation : ∀ p ∈ nodes, ∀ q ∈ nodes, p ≠ q →
    16 ≤ sqDistance (center p) (center q) := by decide +kernel
theorem skew_basis : (4*4*4 : Int) = 64 ∧ (4*1+0*4+0*0 : Int) = 4 := by decide +kernel

theorem packet_merge_clearance : (1-1/2-1/4 : Rat) = 1/4 ∧
    (0 : Rat) < 1/4 := by decide +kernel

-- Exact rational finite-stage branching probabilities.
def success (p : Rat) := p*p*p
def wrong (p q : Rat) := q+p*q+p*p*q

theorem branch_partition (p q d : Rat) (h : p+q+d = 1) :
    success p + wrong p q + wrong p d = 1 := by unfold success wrong; grind

def Y : Rat := success (100/102)
def B : Rat := wrong (100/102) (1/102)
def purity : Rat := Y/(Y+B)

theorem nominal_probabilities : Y = 125000/132651 ∧ B = 7651/265302 ∧
    Y+B+B = 1 := by decide +kernel

theorem nominal_margins : (94/100 : Rat) < Y ∧ purity = 250000/257651 ∧
    (97/100 : Rat) < purity ∧ (3/102 : Rat) = 1/34 := by decide +kernel

def pstar : Rat := (99900/101902)^3
def robustPurity : Rat := pstar/(pstar+(1001/2000)*(1-pstar))

theorem robust_bounds_exact : pstar = 124625374875000/132269020235351 ∧
    robustPurity = 249250749750000000/256902038755711351 ∧
    (94/100 : Rat) < pstar ∧ (97/100 : Rat) < robustPurity := by decide +kernel

-- Independent interval-box corners: 2^9=512 exact rational checks.
def rates : List Rat := [999/10,1001/10]
def errors : List Rat := [999/1000,1001/1000]
def stageCorners : List (Rat × Rat) := rates.flatMap fun a => errors.flatMap fun b =>
  errors.map fun d => (a/(a+b+d),b/(a+b+d))
def triples := stageCorners.flatMap fun x => stageCorners.flatMap fun y =>
  stageCorners.map fun z => (x,y,z)
def heterogeneousY (t : (Rat × Rat) × (Rat × Rat) × (Rat × Rat)) := t.1.1*t.2.1.1*t.2.2.1
def heterogeneousB (t : (Rat × Rat) × (Rat × Rat) × (Rat × Rat)) :=
  t.1.2+t.1.1*t.2.1.2+t.1.1*t.2.1.1*t.2.2.2

theorem corner_count : triples.length = 512 := by decide +kernel
theorem corner_margins : ∀ t ∈ triples, (94/100 : Rat) < heterogeneousY t ∧
    (97/100 : Rat) < heterogeneousY t/(heterogeneousY t+heterogeneousB t) := by decide +kernel

theorem heterogeneous_partition (p₁ q₁ d₁ p₂ q₂ d₂ p₃ q₃ d₃ : Rat)
    (h₁ : p₁+q₁+d₁ = 1) (h₂ : p₂+q₂+d₂ = 1) (h₃ : p₃+q₃+d₃ = 1) :
    p₁*p₂*p₃ + (q₁+p₁*q₂+p₁*p₂*q₃) + (d₁+p₁*d₂+p₁*p₂*d₃) = 1 := by grind

-- Rational quadratic receiver, with c=3 and bound-state b=4,e=10.
def energy (a u lam : Rat) := 8*(a-3)*(a-3)+4*a-10+(2/3)*u*u-lam*u
def minimum (lam : Rat) := 3/2-(3/8)*lam*lam

theorem receiver_square_completion (a u lam : Rat) :
    energy a u lam = 8*(a-11/4)*(a-11/4)+(2/3)*(u-3*lam/4)*(u-3*lam/4)+minimum lam := by
  unfold energy minimum; grind

theorem square_nonnegative (x : Rat) : 0 ≤ x*x := by
  by_cases h : 0 ≤ x
  · exact Rat.mul_nonneg h h
  · have hn : 0 ≤ -x := by grind
    have hh := Rat.mul_nonneg hn hn
    grind

theorem square_positive (x : Rat) (hx : x ≠ 0) : 0 < x*x := by
  by_cases h : 0 < x
```

```
  · exact Rat.mul_pos h h
  · have hn : 0 < -x := by grind
    have hh := Rat.mul_pos hn hn
    grind

theorem receiver_minimum (a u lam : Rat) : minimum lam ≤ energy a u lam := by
  rw [receiver_square_completion]
  have h1 := square_nonnegative (a-11/4)
  have h2 := square_nonnegative (u-3*lam/4)
  have hh1 := Rat.mul_nonneg (show (0 : Rat) ≤ 8 by decide +kernel) h1
  have hh2 := Rat.mul_nonneg (show (0 : Rat) ≤ 2/3 by decide +kernel) h2
  grind

theorem receiver_attainment (lam : Rat) : energy (11/4) (3*lam/4) lam = minimum lam := by
  rw [receiver_square_completion]; grind

theorem receiver_unique (a u lam : Rat) (h : energy a u lam = minimum lam) :
    a = 11/4 ∧ u = 3*lam/4 := by
  rw [receiver_square_completion] at h
  have h1 := square_nonnegative (a-11/4)
  have h2 := square_nonnegative (u-3*lam/4)
  have hh1 := Rat.mul_nonneg (show (0 : Rat) ≤ 8 by decide +kernel) h1
  have hh2 := Rat.mul_nonneg (show (0 : Rat) ≤ 2/3 by decide +kernel) h2
  constructor
  · apply Classical.byContradiction
    intro hn
    have hp := square_positive (a-11/4) (by grind)
    have hpp := Rat.mul_pos (show (0 : Rat) < 8 by decide +kernel) hp
    grind
  · apply Classical.byContradiction
    intro hn
    have hp := square_positive (u-3*lam/4) (by grind)
    have hpp := Rat.mul_pos (show (0 : Rat) < 2/3 by decide +kernel) hp
    grind

theorem receiver_feasible (lam : Rat) (h : 0 ≤ lam ∧ lam ≤ 6) :
    2 ≤ (11/4 : Rat) ∧ (11/4 : Rat) ≤ 6 ∧ -5 ≤ 3*lam/4 ∧ 3*lam/4 ≤ 5 := by
  have h1 := Rat.mul_le_mul_of_nonneg_right h.1 (show (0 : Rat) ≤ 3/4 by decide +kernel)
  have h2 := Rat.mul_le_mul_of_nonneg_right h.2 (show (0 : Rat) ≤ 3/4 by decide +kernel)
  grind

theorem receiver_values : minimum 0 = 3/2 ∧ minimum 4 = -9/2 ∧
    minimum (16/5) = -117/50 ∧ (3/8*(16/5)*(16/5) : Rat) = 96/25 := by decide +kernel

-- Rational rate-law identities only, not a real-valued CTMC convergence proof.
def partition (w v : Rat) : Rat := 1+w+v

theorem basin_normalization (w v : Rat) (hz : partition w v ≠ 0) :
    1/partition w v+w/partition w v+v/partition w v = 1 := by
  have hh := Rat.div_mul_cancel (a := (1 : Rat)) hz
  unfold partition at *
  grind

theorem receiver_detailed_balance (w v nu : Rat) :
    (1/partition w v)*(nu*w/(1+w)) = (w/partition w v)*(nu/(1+w)) ∧
    (1/partition w v)*(nu*v/(1+v)) = (v/partition w v)*(nu/(1+v)) := by
  constructor <;> grind

-- Finite rational Taylor certificates ONLY. Analytic comparison to exp is not
-- formalized. The paper supplies that independent mathematical argument.
def factorial : Nat → Nat | 0 => 1 | n+1 => (n+1)*factorial n
def taylor (x : Rat) (n : Nat) : Rat :=
  ((List.range (n+1)).map fun j => x^j/(factorial j : Rat)).foldr (· + ·) 0
def upper (x : Rat) (n : Nat) : Rat :=
  taylor x n + (x^(n+1)/(factorial (n+1) : Rat))/(1-x/(n+2 : Nat))
def occupancyLower (x y : Rat) := 1/(1+1/taylor x 40+1/taylor y 40)
def occupancyUpper (x y : Rat) := 1/(1+1/upper x 40+1/upper y 40)

theorem absent_rational_enclosure : (15428/100000 : Rat) < 1/(2+upper (3/2) 40) ∧
```

```
    1/(2+taylor (3/2) 40) < (15429/100000 : Rat) := by decide +kernel

theorem alternate_rational_enclosure : (89459/100000 : Rat) < occupancyLower (117/50) (96/25) ∧
    occupancyUpper (117/50) (96/25) < (89461/100000 : Rat) := by decide +kernel

theorem target_rational_enclosure : (98659/100000 : Rat) < occupancyLower (9/2) 6 ∧
    occupancyUpper (9/2) 6 < (98660/100000 : Rat) := by decide +kernel

theorem receiver_contrast_margins :
    (98659/100000-89461/100000 : Rat) = 9198/100000 ∧
    (9/100 : Rat) < 9198/100000-2/10000 ∧
    (98659/100000-15429/100000 : Rat) = 83230/100000 ∧
    (83/100 : Rat) < 83230/100000-2/10000 := by decide +kernel

end Article104

-- Full theorem dependency audit.
#print axioms Article104.execution_composition
#print axioms Article104.execution_preserves_invariant
#print axioms Article104.protected_composition
#print axioms Article104.protected_execution
#print axioms Article104.origin_preserved
#print axioms Article104.balanced_join_degree
#print axioms Article104.unaffected_join_degree
#print axioms Article104.four_endpoint_balance
#print axioms Article104.join_incidence_balance
#print axioms Article104.guarded_join_preserves_degree
#print axioms Article104.decode_encode
#print axioms Article104.actual_prefix_append
#print axioms Article104.code_injective
#print axioms Article104.packet_fresh
#print axioms Article104.peptide_formula_balance
#print axioms Article104.peptide_atom_balance
#print axioms Article104.sixteen_words
#print axioms Article104.all_tetramers_conserve
#print axioms Article104.target_formula
#print axioms Article104.alternate_formula
#print axioms Article104.common_pool
#print axioms Article104.signature_catalogue
#print axioms Article104.signatures_distinguish
#print axioms Article104.full_patch_count
#print axioms Article104.full_patch_edges
#print axioms Article104.full_patch_separation
#print axioms Article104.skew_basis
#print axioms Article104.packet_merge_clearance
#print axioms Article104.branch_partition
#print axioms Article104.nominal_probabilities
#print axioms Article104.nominal_margins
#print axioms Article104.robust_bounds_exact
#print axioms Article104.corner_count
#print axioms Article104.corner_margins
#print axioms Article104.heterogeneous_partition
#print axioms Article104.receiver_square_completion
#print axioms Article104.square_nonnegative
#print axioms Article104.square_positive
#print axioms Article104.receiver_minimum
#print axioms Article104.receiver_attainment
#print axioms Article104.receiver_unique
#print axioms Article104.receiver_feasible
#print axioms Article104.receiver_values
#print axioms Article104.basin_normalization
#print axioms Article104.receiver_detailed_balance
#print axioms Article104.absent_rational_enclosure
#print axioms Article104.alternate_rational_enclosure
#print axioms Article104.target_rational_enclosure
#print axioms Article104.receiver_contrast_margins
```

Recorded kernel dependency audit

The successful build produced the following per-theorem audit. There were no compilation errors or unproved obligations.

```
'Article104.execution_composition' does not depend on any axioms
'Article104.execution_preserves_invariant' does not depend on any axioms
'Article104.protected_composition' does not depend on any axioms
'Article104.protected_execution' does not depend on any axioms
'Article104.origin_preserved' does not depend on any axioms
'Article104.balanced_join_degree' depends on axioms: [propext, Quot.sound]
'Article104.unaffected_join_degree' depends on axioms: [propext, Quot.sound]
'Article104.four_endpoint_balance' depends on axioms: [propext, Quot.sound]
'Article104.join_incidence_balance' depends on axioms: [propext, Quot.sound]
'Article104.guarded_join_preserves_degree' depends on axioms: [propext, Quot.sound]
'Article104.decode_encode' depends on axioms: [propext]
'Article104.actual_prefix_append' depends on axioms: [propext]
'Article104.code_injective' depends on axioms: [propext]
'Article104.packet_fresh' depends on axioms: [propext, Quot.sound]
'Article104.peptide_formula_balance' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.peptide_atom_balance' depends on axioms: [propext, Quot.sound]
'Article104.sixteen_words' does not depend on any axioms
'Article104.all_tetramers_conserve' does not depend on any axioms
'Article104.target_formula' does not depend on any axioms
'Article104.alternate_formula' does not depend on any axioms
'Article104.common_pool' does not depend on any axioms
'Article104.signature_catalogue' does not depend on any axioms
'Article104.signatures_distinguish' does not depend on any axioms
'Article104.full_patch_count' does not depend on any axioms
'Article104.full_patch_edges' does not depend on any axioms
'Article104.full_patch_separation' does not depend on any axioms
'Article104.skew_basis' does not depend on any axioms
'Article104.packet_merge_clearance' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.branch_partition' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.nominal_probabilities' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.nominal_margins' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.robust_bounds_exact' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.corner_count' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.corner_margins' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.heterogeneous_partition' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_square_completion' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.square_nonnegative' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.square_positive' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_minimum' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_attainment' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_unique' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_feasible' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_values' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.basin_normalization' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_detailed_balance' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.absent_rational_enclosure' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.alternate_rational_enclosure' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.target_rational_enclosure' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article104.receiver_contrast_margins' depends on axioms: [propext, Classical.choice, Quot.sound]
```

Publication description

The supported description is “49 kernel-checked theorems in a partial finite/rational formalization of Article 104.” The stronger descriptions “the entire paper is formally proved” and “chemical existence has been machine-verified” are not supported by this artifact.