Article 103: Lean 4 Formal Proof Supplement

From Fibrated Chrysene Networks to Molecular Machines

Charles D. Schaper

Volume 1, Number 5; companion to the manuscript beginning at page 1975
Original verification: 21 October 2026; independently rerun for publication, November 2026

Abstract

This supplement contains a separately compiled Lean 4 module with 47 kernel-checked theorems for the structural and exact rational core of Article 103. Generic results include interface composition, invariant preservation, a coding representative theorem, a historical-recovery obstruction, completion of the gate energy square, attainment and uniqueness of its rational global minimum, rational occupancy bounds, detailed-balance identities, and control-reset work. Exact certificates cover the sixteen-core reference geometry, edge counts, energy values, Taylor partial sums, and success margins. This is a partial formalization, not a machine-checked proof of every analytic theorem or of molecular manufacture. Its precise boundary is recorded below.

1 Executed status and reproduction

The accompanying `Article103.lean` was compiled successfully, with process exit status zero, using the official Lean 4.24.0 Linux release. The exact compiler identifier is:

```
Lean 4.24.0, x86_64-unknown-linux-gnu, Release
797c613eb9b6d4ec95db23e3e00af9ac6657f24b
```

Only `Std` and `Init.Data.Rat.Lemmas` are imported. No Mathlib download is needed. The pinned `lean-toolchain` selects `leanprover/lean4:v4.24.0`. From the source directory run:

```
sh build.sh
# Or, with an explicit executable:
LEAN=/path/to/lean-4.24.0-linux/bin/lean sh build.sh
```

The build prints the compiler version, produces `Article103.olean`, and prints a dependency audit for every theorem. The supplied `lean-build.log` records the completed run. Compiling this PDF in Overleaf does not execute Lean; the Lean run and LaTeX run are separate checks.

There are no admitted proofs, custom axioms, or uses of `native_decide`. Exact finite checks use `decide +kernel`; algebraic proof search uses `grind`, whose resulting proof terms are checked by the kernel. The complete audit reports only the standard foundational axioms `propext`, `Classical.choice`, and `Quot.sound`, or no axioms. In particular, it reports neither `sorryAx` nor an external-computation axiom. Classical representative selection is intentionally noncomputable; it is not a synthesis algorithm.

Source integrity (SHA-256):

```
732d2e0aa06a5dca95c8ec07ce62d7419bb8e92fc5ee72287de49c4a51fef14a
```

The source file is authoritative for copy-and-run reproduction. Mathematical glyphs in a PDF may be substituted during copying; use the bundled raw UTF-8 file rather than relying on PDF extraction.

2 Correspondence with the revised manuscript

1. **Finite structural restriction and symmetry.** `restriction_injective` proves that restricting an injective address map to a subtype preserves injectivity. `distinguishability_obstruction` proves that different observed outputs exclude a specified observation-preserving symmetry relation. The invariance hypothesis is supplied explicitly. These theorems do not formalize the full fibrated network axioms, physical port covariance, or a symmetry theorem for continuous-time path laws.

2. **Compatible finite executions and code–product interfaces.** `Execution` records a finite chain of allowed transitions with the same intermediate state. Composition and invariant preservation are proved by induction. `decoder_execution` specializes the invariant to code–product consistency, assuming that every permitted step preserves that consistency. It does not create a chemical transition from a decoder identity.

3. **Coding lift and the historical distinction.** For an explicitly covering finite code list, `finite_code_representative` chooses a code in that list decoding to each accepted product. No uniqueness is claimed. `history_not_recoverable` proves that a decoder with two distinct codes for one product cannot have a product-only encoder recovering both original codes. This distinguishes a representative reverse encoding from recovery of an actual production history.

4. **Sixteen-core reference witness.** The duplicate-free list `nodes` has sixteen entries, with sixteen oriented-once lateral edges and eight oriented-once stacking edges. The reference centers are $(12b + 4i + j, 4j, 4k)$ for binary indices. Exhaustive kernel reduction verifies squared center distance at least 16 for distinct listed nodes. The basis determinant is 64 and the lateral dot product is 4. This is a reference-configuration check at aperture $a = 5$, not a universal certificate for every continuous configuration.

5. **Routing-map identities and clearance budgets.** The rational piecewise-affine map `hmap` fixes the left region, translates the right region, matches its branch endpoints, and sends $17/2$ to $6 + a/2$. Rational scalar bounds verify $a/5 \geq 2/5$, substrate clearance $a/2 - 1/4 \geq 3/4$, and the positive tube budget $2/5 - 3/40 - 1/4 = 3/40$. No theorem here constructs exterior connector arcs, exit anchors or sleeves, proves a global bi-Lipschitz estimate, or verifies disjoint continuous tubes. These remain explicit geometric hypotheses and manuscript arguments.

6. **Generic quadratic mechanics over the rationals.** For all rational a, c, b, E, the module proves

$$8(a - c)^2 + ba - E = 8(a - (c - b/16))^2 + bc - E - b^2/32.$$

   It proves nonnegativity and strict positivity of rational squares, then the attained unique global minimum $a = c - b/16$. It also checks occupied-minimizer feasibility for $3 \leq c \leq 5$, all four endpoint energies, and the registry energy gap $6r$. These are genuine universally quantified rational results, not a finite parameter sweep. Extension to real variables, differentiability, Hessian stability and neighborhood robustness are not machine checked here.

7. **Probability arithmetic.** Four exact finite Taylor sums exceed 80, 4, 33, and 10000, respectively. The module does *not* define or prove facts about the real exponential. It separately proves that rational weights satisfying $w > 80$, $0 \leq v < 1/4$ give $w/(1 + w + v) > 0.98$, and that $0 \leq w < 1/33$, $0 \leq v < 1/10000$ give $1/(1 + w + v) > 0.97$. The real Taylor remainder and the link from Boltzmann weights to these hypotheses remain analytic obligations. The exact cycle product is $0.95040501 > 0.95$; the interpretation as a sequential success bound still requires the conditional-probability and execution hypotheses in the manuscript.

8. **Kinetic and selection normalization.** The module proves rational normalization of the three-state occupancy weights for nonzero normalizer, both star-edge detailed-balance identities, and finite accepted-law normalization. Positivity is required for a probability/rate interpretation, and list sums count multiplicities. These identities alone do not prove irreducibility, convergence, a spectral gap, a stopping-time distribution, or a leakage bound.

9. **Control and energy bookkeeping.** The generic reset identity records the change in mechanical energy caused by replacing c_- with c_+ at fixed aperture and fixed occupancy. It is separate from occupancy-transition energy. The rest-control identity and algebraic path-lift identity are checked for nonzero mobility. No ODE existence, differentiability, finite-time controller implementation, actuator feasibility, occupancy-hold capability, or energy-supply closure is inferred from those identities.

3 What is deliberately not certified

This companion does not formalize real constrained minimization on general compact configuration spaces, the implicit-function theorem, differentiable energy balance, continuous routing and deformation, sampled-path Lipschitz certificates, spectral mixing and stochastic clock changes, real-valued robustness neighborhoods, strong-convexity residual estimates, retention-tube topology, or the general peptide leakage inequality. These remain part of the manuscript's ordinary mathematical proofs under stated assumptions. It also does not prove the realizability of field couplings, stable substituted-chrysene assemblies, alpha-helix nucleation, PCET chemistry, polypeptide synthesis, a helicase, autonomous self-assembly, or experimental performance.

The formal certificate should therefore be cited as "kernel-checked structural and rational core," not as "a complete Lean proof of Article 103." Exact algebra confirms internal consistency of the declared reduced construction; it cannot establish that the construction is realized in a particular chemical system.

4 Complete Lean source

The following listing is the complete source, including all theorem dependency audits. Line wrapping is typographical and does not alter the accompanying raw file.

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

/-!
Article 103, a PARTIAL formalization of the declared reduced model.
Lean 4.24.0 / Std only. All numerical constants are dimensionless.
No theorem below certifies molecular synthesis or physical calibration.
Real exponentials, continuous geometric routing, CTMC convergence,
and real differential equations are not formalized here.
-/
namespace Article103

-- Generic finite-interface structure, not a chemical reaction assertion.
def Injective {A B : Type} (f : A → B) := ∀ {a b}, f a = f b → a = b

theorem restriction_injective {A B : Type} (f : A → B)
    (hf : Injective f) (P : A → Prop) :
    Injective (fun x : {a // P a} => f x.val) := by
  intro a b h
  exact Subtype.eq (hf h)

theorem symmetry_no_contrast {X Y : Type} (g : X → X) (O : X → Y)
    (invariant : ∀ x, O (g x) = O x) (x : X) : O (g x) = O x := invariant x

-- Nonidentical observations exclude membership in the specified symmetry orbit.
theorem distinguishability_obstruction {X Y : Type} (g : X → X) (O : X → Y)
    (invariant : ∀ x, O (g x) = O x) (x y : X) (different : O y ≠ O x) :
    y ≠ g x := by
  intro h
  exact different (h ▸ invariant x)

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)

-- A joint-production interface needs decoder compatibility at EVERY step.
theorem decoder_execution {C P : Type} (D : C → P)
    (step : (C × P) → (C × P) → Prop)
    (compatible : ∀ x y, step x y → D x.1 = x.2 → D y.1 = y.2)
    {x y : C × P} (h : Execution step x y) (hx : D x.1 = x.2) : D y.1 = y.2 :=
  execution_preserves_invariant (fun z => D z.1 = z.2) compatible h hx
```

```
55
56 -- A finite enumeration yields a recorded representative, not historical identity.
57 theorem finite_code_representative {C P : Type} (codes : List C) (D : C → P)
58     (accepted : P → Prop)
59     (covered : ∀ p, accepted p → ∃ c, c ∈ codes ∧ D c = p) :
60     ∃ encode : {p // accepted p} → C,
61       ∀ p, encode p ∈ codes ∧ D (encode p) = p.val := by
62   classical
63   exact ⟨fun p => Classical.choose (covered p.val p.property),
64     fun p => Classical.choose_spec (covered p.val p.property)⟩
65
66 theorem history_not_recoverable {C P : Type} (D : C → P) (c d : C)
67     (same : D c = D d) (distinct : c ≠ d) :
68     ¬ ∃ encode : P → C, encode (D c) = c ∧ encode (D d) = d := by
69   rintro ⟨encode, hc, hd⟩
70   apply distinct
71   exact hc.symm.trans ((congrArg encode same).trans hd)
72
73 -- Exact finite geometry of the two blocks at reference aperture a = 5.
74 abbrev Node := Fin 2 × Fin 2 × Fin 2 × Fin 2
75 abbrev Vec := Int × Int × Int
76
77 def center (n : Node) : Vec :=
78   let b : Int := n.1.val
79   let i : Int := n.2.1.val
80   let j : Int := n.2.2.1.val
81   let k : Int := n.2.2.2.val
82   (12*b + 4*i+j, 4*j, 4*k)
83
84 def squaredDistance (p q : Vec) : Int :=
85   (p.1-q.1)*(p.1-q.1) + (p.2.1-q.2.1)*(p.2.1-q.2.1) +
86   (p.2.2-q.2.2)*(p.2.2-q.2.2)
87
88 def nodes : List Node := (List.finRange 2).flatMap fun b =>
89   (List.finRange 2).flatMap fun i => (List.finRange 2).flatMap fun j =>
90   (List.finRange 2).map fun k => (b,i,j,k)
91
92 def lateral (p q : Node) : Bool :=
93   p.1 == q.1 && p.2.2.2 == q.2.2.2 &&
94   ((p.2.1.val + 1 == q.2.1.val && p.2.2.1 == q.2.2.1) ||
95    (p.2.2.1.val + 1 == q.2.2.1.val && p.2.1 == q.2.1))
96 def stacked (p q : Node) : Bool :=
97   p.1 == q.1 && p.2.1 == q.2.1 && p.2.2.1 == q.2.2.1 &&
98   p.2.2.2.val + 1 == q.2.2.2.val
99 def edgeCount (edge : Node → Node → Bool) : Nat :=
100   (nodes.flatMap fun p => nodes.filter fun q => edge p q).length
101
102 theorem sixteen_nodes : nodes.length = 16 := by decide +kernel
103 theorem nodes_distinct : nodes.Nodup := by decide +kernel
104 theorem lateral_edges : edgeCount lateral = 16 := by decide +kernel
105 theorem stacking_edges : edgeCount stacked = 8 := by decide +kernel
106 theorem reference_separation : ∀ p ∈ nodes, ∀ q ∈ nodes, p ≠ q →
107     16 ≤ squaredDistance (center p) (center q) := by decide +kernel
108
109 theorem reference_basis :
110     (4*(4*4-0*0)-1*(0*4-0*0)+0*(0*0-4*0) : Int) = 64 ∧
111     (4*1+0*4+0*0 : Int) = 4 := by decide +kernel
112
113 -- Rational piecewise affine routing map. This is not a proof of routing existence.
```

```
def hmap (a x : Rat) : Rat :=
  if x ≤ 6 then x else if x ≤ 11 then 6+(a/5)*(x-6) else x+a-5

theorem hmap_left (a x : Rat) (hx : x ≤ 6) : hmap a x = x := by
  simp [hmap, hx]
theorem hmap_right (a x : Rat) (hx : 11 < x) : hmap a x = x+a-5 := by
  have h6 : ¬ x ≤ 6 := by grind
  have h11 : ¬ x ≤ 11 := by grind
  simp [hmap, h6, h11]
theorem hmap_midpoint (a : Rat) : hmap a (17/2) = 6+a/2 := by
  unfold hmap
  split <;> (try contradiction) <;> grind

theorem hmap_boundary_matches (a : Rat) :
    6+(a/5)*(6-6) = 6 ∧ 6+(a/5)*(11-6) = 11+a-5 := by grind

theorem routing_scale_budget (a : Rat) (ha : 2 ≤ a) : 2/5 ≤ a/5 := by
  have hp := Rat.mul_le_mul_of_nonneg_right ha (show (0 : Rat) ≤ 1/5 by decide +kernel)
  grind

theorem substrate_clearance_budget (a : Rat) (ha : 2 ≤ a) :
    3/4 ≤ a/2-1/4 := by
  have hp := Rat.mul_le_mul_of_nonneg_right ha (show (0 : Rat) ≤ 1/2 by decide +kernel)
  grind

theorem tube_clearance_budget : (2/5-3/40-1/4 : Rat) = 3/40 ∧
    (0 : Rat) < 3/40 := by decide +kernel

-- Generic rational quadratic energy, independent of any molecular interpretation.
def energy (a c b E : Rat) : Rat := 8*(a-c)*(a-c)+b*a-E
def minimum (c b E : Rat) : Rat := b*c-E-b*b/32

theorem complete_square (a c b E : Rat) :
    energy a c b E = 8*(a-(c-b/16))*(a-(c-b/16))+minimum c b E := 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 quadratic_minimum (a c b E : Rat) : minimum c b E ≤ energy a c b E := by
  rw [complete_square]
  have h := square_nonnegative (a-(c-b/16))
  grind

theorem quadratic_minimum_attained (c b E : Rat) :
    energy (c-b/16) c b E = minimum c b E := by
  rw [complete_square]
```

```
  grind

theorem quadratic_minimum_unique (a c b E : Rat)
    (h : energy a c b E = minimum c b E) : a = c-b/16 := by
  apply Classical.byContradiction
  intro hn
  have hd : a-(c-b/16) ≠ 0 := by grind
  have hp := square_positive (a-(c-b/16)) hd
  rw [complete_square] at h
  grind

theorem occupied_minimum_feasible (c : Rat) (hc : 3 ≤ c ∧ c ≤ 5) :
    11/4 ≤ c-4/16 ∧ c-4/16 ≤ 5 ∧ 2 ≤ c-4/16 ∧ c-4/16 ≤ 6 := by grind

theorem gate_energy_values :
    minimum 3 4 16 = -9/2 ∧ minimum 3 4 10 = 3/2 ∧
    minimum 5 4 16 = 7/2 ∧ minimum 5 4 10 = 19/2 := by decide +kernel

theorem registry_energy_contrast (c r : Rat) :
    minimum c 4 10 - minimum c 4 (10+6*r) = 6*r := by
  unfold minimum
  grind

-- Exact rational Taylor PARTIAL SUMS, not the real exponential function.
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

theorem capture_taylor : taylor (9/2) 7 = 1178747/14336 ∧
    80 < taylor (9/2) 7 := by decide +kernel
theorem competitor_taylor : taylor (3/2) 3 = 67/16 ∧
    4 < taylor (3/2) 3 := by decide +kernel
theorem release_taylor : taylor (7/2) 9 = 175207859/5308416 ∧
    33 < taylor (7/2) 9 := by decide +kernel
theorem release_competitor_taylor : taylor (19/2) 11 = 821287264323721/81749606400 ∧
    10000 < taylor (19/2) 11 := by decide +kernel

theorem capture_rational_margin : (80/(80+1+1/4) : Rat) = 64/65 ∧
    (49/50 : Rat) < 64/65 := by decide +kernel
theorem release_rational_margin : (1/33+1/10000 : Rat) < 3/97 ∧
    (97/100 : Rat) < 1/(1+1/33+1/10000) := by decide +kernel

theorem completed_cycle_margin :
    ((98/100-1/10000)*(97/100-1/10000) : Rat) = 95040501/100000000 ∧
    (95/100 : Rat) < (98/100-1/10000)*(97/100-1/10000) := by decide +kernel

-- Rational algebra of the star-shaped reversible rate completion.
-- Positive weights/frequencies are needed for physical probabilistic meaning.
def Z (w v : Rat) : Rat := 1+w+v

theorem occupancy_normalization (w v : Rat) (hz : Z w v ≠ 0) :
    1/Z w v + w/Z w v + v/Z w v = 1 := by
  have h := Rat.div_mul_cancel (a := (1 : Rat)) hz
  unfold Z at *
  grind

```

```
theorem target_detailed_balance (w v nu : Rat) :
    (1/Z w v)*(nu*w/(1+w)) = (w/Z w v)*(nu/(1+w)) := by grind

theorem competitor_detailed_balance (w v nu : Rat) :
    (1/Z w v)*(nu*v/(1+v)) = (v/Z w v)*(nu/(1+v)) := by grind

-- Finite sums and accepted-law normalization, assuming nonzero total mass.
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 accepted_law_normalizes {X : Type} (xs : List X) (weights : X → Rat)
    (hn : mass xs weights ≠ 0) :
    mass xs (fun x => (1/mass xs weights)*weights x) = 1 := by
  rw [mass_scale]
  exact Rat.div_mul_cancel hn


-- These bounds assume rational weights with the specified inequalities.
-- Linking such hypotheses to real Boltzmann weights is outside this module.
theorem capture_probability_bound (w v : Rat) (hw : 80 < w) (hv : 0 ≤ v ∧ v < 1/4) :
    49/50 < w/(1+w+v) := by
  have hz : 0 < 1+w+v := by grind
  have hv4 : v*4 < 1 := (Rat.lt_div_iff (by decide +kernel)).mp hv.2
  rw [Rat.lt_div_iff hz]
  have he : (49/50 : Rat)*(1+w+v) = (49*(1+w+v))/50 := by grind
  rw [he, Rat.div_lt_iff (by decide +kernel)]
  grind
theorem release_probability_bound (w v : Rat) (hw : 0 ≤ w ∧ w < 1/33) (hv : 0 ≤ v ∧ v < 1/10000) :
    97/100 < 1/(1+w+v) := by
  have hz : 0 < 1+w+v := by grind
  have hw33 : w*33 < 1 := (Rat.lt_div_iff (by decide +kernel)).mp hw.2
  have hv10000 : v*10000 < 1 := (Rat.lt_div_iff (by decide +kernel)).mp hv.2
  rw [Rat.lt_div_iff hz]
  have he : (97/100 : Rat)*(1+w+v) = (97*(1+w+v))/100 := by grind
  rw [he, Rat.div_lt_iff (by decide +kernel)]
  grind

-- Control reset work is separate from an occupancy energy jump.
theorem reset_work_identity (a cm cp b E : Rat) :
    energy a cp b E - energy a cm b E =
      8*((a-cp)*(a-cp)-(a-cm)*(a-cm)) := by
  unfold energy
  grind

theorem rest_control_force (a b : Rat) : 16*(a-(a+b/16))+b = 0 := by grind

-- Algebraic path lift only: no differentiability or ODE theorem is asserted.
theorem path_lift_identity (a velocity mu b : Rat) (hm : mu ≠ 0) :
    -mu*(16*(a-(a+b/16+velocity/(16*mu)))+b) = velocity := by
  have hi := Rat.div_mul_cancel (a := velocity) hm
  grind
```

```
291
292
293 end Article103
294
295 -- Full dependency audit of every theorem in this module.
296 #print axioms Article103.restriction_injective
297 #print axioms Article103.symmetry_no_contrast
298 #print axioms Article103.distinguishability_obstruction
299 #print axioms Article103.execution_composition
300 #print axioms Article103.execution_preserves_invariant
301 #print axioms Article103.decoder_execution
302 #print axioms Article103.finite_code_representative
303 #print axioms Article103.history_not_recoverable
304 #print axioms Article103.sixteen_nodes
305 #print axioms Article103.nodes_distinct
306 #print axioms Article103.lateral_edges
307 #print axioms Article103.stacking_edges
308 #print axioms Article103.reference_separation
309 #print axioms Article103.reference_basis
310 #print axioms Article103.hmap_left
311 #print axioms Article103.hmap_right
312 #print axioms Article103.hmap_midpoint
313 #print axioms Article103.hmap_boundary_matches
314 #print axioms Article103.routing_scale_budget
315 #print axioms Article103.substrate_clearance_budget
316 #print axioms Article103.tube_clearance_budget
317 #print axioms Article103.complete_square
318 #print axioms Article103.square_nonnegative
319 #print axioms Article103.square_positive
320 #print axioms Article103.quadratic_minimum
321 #print axioms Article103.quadratic_minimum_attained
322 #print axioms Article103.quadratic_minimum_unique
323 #print axioms Article103.occupied_minimum_feasible
324 #print axioms Article103.gate_energy_values
325 #print axioms Article103.registry_energy_contrast
326 #print axioms Article103.capture_taylor
327 #print axioms Article103.competitor_taylor
328 #print axioms Article103.release_taylor
329 #print axioms Article103.release_competitor_taylor
330 #print axioms Article103.capture_rational_margin
331 #print axioms Article103.release_rational_margin
332 #print axioms Article103.completed_cycle_margin
333 #print axioms Article103.occupancy_normalization
334 #print axioms Article103.target_detailed_balance
335 #print axioms Article103.competitor_detailed_balance
336 #print axioms Article103.mass_scale
337 #print axioms Article103.accepted_law_normalizes
338 #print axioms Article103.capture_probability_bound
339 #print axioms Article103.release_probability_bound
340 #print axioms Article103.reset_work_identity
341 #print axioms Article103.rest_control_force
342 #print axioms Article103.path_lift_identity
```