ACF 88 — Lean 4 Proof Dossier

A Finite Axiomatic Framework for the Interior Chrysene Lattice and Protocell Organization

Charles D. Schaper

Annals of the Chrysene Formalism, Vol. 1, Issue 4, pp. 1778–1807

Audited companion: 31 October 2026 Publication formatting: 3 November 2026

Kernel Certification and Scope

Item	Audited release result
Canonical proof	`Article85.lean`
Pinned Lean version	`4.24.0`; bundled libraries only
Authored theorems and lemmas	129
Namespace declarations audited	682
Unresolved proof placeholders	**None**
Custom or native-evaluation proof axioms	**None accepted**
Foundational axiom union	`propext`, `Classical.choice`, `Quot.sound`
Coverage	Partial; exclusions are stated below

1 Scope and proof status

This is a **partial, kernel-checked formal proof companion** to Article 88, *A Finite Axiomatic Framework for the Interior Chrysene Lattice and Protocell Organization*, using its controlling 11 October 2026 release, originally identified as Article 85. The original audited mathematical body and proof code are unchanged. Publication formatting and a contextual literature appendix (Appendix G) have been added. The qualification "partial" concerns coverage, not unfinished Lean proofs: every included declaration is checked, and no unresolved proof placeholder or new proof axiom is admitted.

The complete source is one `Article85.lean` file, pinned to official Lean `4.24.0`. It imports only the bundled Lean libraries, with no Mathlib download, external theorem oracle, precompiled project module, or native evaluation proof shortcut. The digital PDF carries the exact Unicode source inside its complete recoverable audit runtime.

The frozen source contains **129 named authored theorems and lemmas**. A fresh environment audit covers **all 682 declarations** in its `Article85` namespace, including generated constructors and helpers. These counts are not counts of distinct manuscript theorems. The observed foundational dependencies are exactly the union `propext`, `Classical.choice` and `Quot.sound`; many individual declarations have no axiom dependencies. No `sorryAx`, custom proof axiom, or native-evaluation axiom is accepted.

The source release archive SHA-256 is

```
f5956f59ecd009368e784708bb44dc8860
70421ec1981601fc8143aeaeb49ade
```

The bundled `SOURCE_PIN.json` records the reviewed manuscript files. Formal assumptions are explicit theorem parameters or structure fields, not hidden replacement axioms asserting manuscript conclusions.

For reproducibility, executable code, Lean namespaces, recovery archives, historical filenames and source hashes keep their original Article 85 identities. The frozen `SOURCE_PIN.json` governs coverage of the original audited statements; the new contextual Appendix G is outside that audit.

2 The mathematical kernels

2.1 Recognition is a general theorem

Let G be a symmetric graph with an indicated root and a walk between every pair of vertices. An edge increment $s(x, y) \in \mathbb{Z}^3$ satisfies $s(y, x) = -s(x, y)$. For a walk p write $I(p)$ for its sum of increments. The proof establishes

$$(\exists a : V \to \mathbb{Z}^3, \quad a(y) - a(x) = s(x, y) \text{ on edges}) \quad \Longleftrightarrow \quad I(p) = 0 \text{ for every closed walk } p.$$

It constructs coordinates from root walks and proves $I(p) = a(y) - a(x)$ for every walk from x to y. Two coordinate maps differ by one translation. A coordinate map is injective exactly when a zero-increment walk has equal endpoints. Finally, when edge increments are signed coordinate-unit vectors, adding every available coordinate-neighbor incidence gives precisely an induced lattice realization. This proves the logical core of the finite recognition theorem without assuming its conclusion in a structure field. Finiteness, molecular decorations and tensor data are not needed for this graph theorem and are not asserted by it.

2.2 Undefined operations remain undefined

A partial operation is encoded by a function into `Option`: `none` is a failed guard and `some y` a successful result. For an encoding E, the one-step identity has the form

$$T_u(E(x)) = \mathrm{map}(E)(f_u(x)).$$

This equality includes the domain equivalence, rather than comparing outputs only when both happen to be defined. Induction extends it to every finite word. Quotient maps have the corresponding identity with the projection on returned values. The formalized counterexamples show why disjoint changed coordinates alone do not preserve another operation's guard, and why a correct single decoded step does not guarantee a correct second step without image invariance.

2.3 Addressing and preparation have distinct types

For a six-bit word u, complement tag a, and reversal tag d, the reference-normal word is $F^a R^d u$. The resident carrier remains AAA. The selected normalized words 100110 and 100111 encode GCG and GCU and prepare respectively $(m, r) = (1, 1)$ and $(0, 1)$. Their preparation maps are constant state assignments. They are not the later operation $u(m, r) = (m, r \oplus m)$, which preserves the control bit m. Neither the normalization theorem nor that register operation creates a residue-selection or covalent joining mechanism. The printed definitions retain these type boundaries.

3 Correspondence with the article

Article statement	Formal scope
Theorem 2.2	General walks in a connected symmetric directional graph: zero periods if and only if coordinates exist, uniqueness up to translation, zero-increment separation if and only if injectivity, and the induced-neighbor completeness criterion. The argument is parametric, not an enumeration of the 27-cell box.
Propositions 2.4 and 4.1	Actual-domain partial translation identities and a counterexample to unrestricted composition. These do not identify the maximal domain of a direct translation with that of a composite.
Theorem 7.3	Option-valued partial maps encode undefined guards. Intertwining identities extend to arbitrary finite input words, including exact legality/reflection and observations. Encoded and quotient maps compose.
Theorem 8.3	Finite-operation invariant transfer, guarded inventory arithmetic and a product-coordinate sufficient noninterference specialization. This is not a formalization of every possible read/write footprint system.
Theorems 9.1–9.2	Finite register encoding/decoding and typed preparations; rational path/cycle terminal identities and scalar Schur reduction. Contextual replacement is the reduced-equation consequence of equal terminal laws and unique solutions, not the entire passive-network positivity and matrix-elimination theorem.
Theorem 10.1; Section 10	Fixed-AAA reference normalization and selected initialization. State-valued initialization and operation-valued preparation dispatch are distinct; both are distinguished from the subsequent register update. Six-bit and finite address claims are finite specializations, not claims about genetic expression.
Proposition 11.1	Shared-read compatibility as the existence and uniqueness condition for a product update; additive cancellation in its stated finite algebraic specialization. No continuous fiber-product ODE theorem.
Native/material/count identities	Exact finite/rational arithmetic, native inventories, and the distinction between atom ancestry and current protected bonds. Carbon or element balance is not reinterpreted as a full atom map or admissible chemical pathway.

The exact declaration-level map is in `COVERAGE.json`, and `THEOREMS.json` lists every authored theorem printed by the source. The 36 named statements of the manuscript were separately reviewed for coverage and exclusions. In particular, the 36/20 coarser address counts are finite list computations of orbit-normal keys, not a separate Lean quotient-cardinality construction. Native count and displacement certificates do not prove the complete real embedding or maximal point group. The exact rational availability, threshold, water, energy and capsule inequalities are supporting arithmetic only. The general boundary lower bound, absence of nonzero total finite translations, and Hilbert-space partial-shift identities are not formalized here.

The current-material kernel stores immutable element and origin maps separately from current

bond orders. It proves that editing outside protected pairs leaves those bonds unchanged and that arbitrary legal words preserve them when every step does. It does not assert a complete valence-admissible split/join construction from the balanced packet formulas.

3.1 Explicit exclusions

This file does not prove the full F1–F12 existence theorem as one Lean structure, the generated complex matrix algebra, standard-form commutant theorem, trace-Dirac norm/distance formula, native-generator Dirac theorem, general geometric spectral-distance supremum, causal response convergence/holomorphy, or continuous quantum evolution. It does not prove real ODE existence/uniqueness, the hyperbolic-tangent solution, capsule enclosure, or the whole hybrid threshold theorem for every admissible attenuation function. Rational prerequisites for those arguments are not relabeled as their analytic conclusions. No physical molecular realization, actual atom-resolved productive block, reaction mechanism, experimentally calibrated rate, biological translation, chirality selection or functioning protocell is established.

4 Statement-by-statement coverage

The following inventory covers every named theorem or proposition in the controlling manuscript. "Excluded" means that the manuscript statement is not claimed as a Lean theorem here; arithmetic examples mentioned in the same row do not change that status. Section and statement numbering is that of the unchanged 11 October manuscript.

Manuscript statement	Exact Lean scope
2.2 Recognition and uniqueness	**General graph proof**. Full coordinate-existence/zero-period equivalence, translation uniqueness, walk separation/injectivity equivalence, and induced incidence; arbitrary connected symmetric integer-labelled graphs, specialized by signed-unit edge condition.
2.4 Boundary count and partial translations	**Partial**. Translation composition on actual intermediate domain and explicit missing-intermediate counterexample; finite box boundary count. No general finite-set boundary or no-total-translation theorem.
4.1 Finite boundary and partial shifts	**Partial**. Actual-domain translation identity, explicit failed-intermediate counterexample and finite boundary arithmetic; not general operator adjoints/projections.
4.2 The finite generated algebra	**Excluded**. No complex C-star algebra generation formalization.
4.3 Finite standard-form duality	**Excluded**. No general complex standard-form commutant or antiunitarity theorem.
4.4 A nondegenerate finite spectral operator	**Excluded**. No operator-norm or state-space supremum proof.
4.6 Native-generator spectral structure	**Excluded**. No native-generator Dirac theorem.
4.7 Separation and finite admissible paths	**Excluded**. No finite-dimensional norm compactness/state separation or general shortest-path proof.
4.8 Finite directional resolvent identities	**Partial scalar algebra only**. Scalar rational Schur elimination, not finite matrix resolvent theorem or norm bound.
4.9 Finite causal response	**Excluded**. No convergence, holomorphy, spectral-integral derivation, or general response bound.
5.1 Finite box counts	**Finite integer kernels**. Native/box cardinalities, boundary and inventory; no geometry over real square roots.

Manuscript statement	Exact Lean scope
5.2 Existence of a nontrivial finite ICL	**Partial witness kernels only**. No single Lean structure satisfying all F1-F12; finite native, component and address facts are supporting results.
5.3 Deterministic finite Hamiltonian evolution	**Excluded**. No matrix ODE/exponential unitary evolution theorem.
6.1 Geometric spectral distance	**Excluded**. No operator norm/supremum theorem.
7.3 Composition of realizations	**General dynamical core**. Exact Option-valued intertwining, composition, legal arbitrary words, encoded invariant-image output, quotient lifting from any state above a source, and observation traces. Observation compatibility is explicit.
7.4 Strictness of the distinctions	**Selected counterexample kernels**. Ancestry does not fix current bonds; no complete equivalence hierarchy formalization.
8.1 Gluing criterion	**Excluded**. No molecular/tensor gluing theorem.
8.3 Composition and noninterference	**General predicate/quantity core; restricted commutation**. Arbitrary legal-word predicate and quantity preservation; product-coordinate guards give a sufficient noninterference specialization. Arbitrary read/write footprint systems are not formalized.
8.5 Finite extension	**Excluded**. No successive fresh-atom conservative-extension theorem.
8.7 Exact helical geometry	**Excluded**. No trigonometric helix geometry proof.
9.1 Selected structural and dynamical realization	**Finite action/error-correction kernels**. Three/four-bit round trips, single-error correction, disjoint balls, action and selected preparation identities. Ordered geometric ports and molecular interpretation are not formalized.
9.2 Contextual terminal equivalence	**Conditional rational reduced-law kernel**. All-rational path/cycle currents, midpoint balance/uniqueness, scalar Schur elimination; equal reduced module laws plus unique common context solutions imply equal external observations. Does not derive Laplacian positive definiteness or general matrix Schur elimination.
10.1 Normal form and genetic action	**Normal form and selected dispatch**. Fixed AAA type, two involutions and their composition, canonical orbit representative, normal-form equivalence, codon bijection and transported reversal; typed selected preparations and nonidentity later operation; exact six-bit enumeration gives 64 normalized words, 256 tagged addresses and 36/20 canonical class counts. Biological output/translation excluded.
11.1 Discrete compatibility	**General compatibility core; finite flux specialization**. Heterogeneous dependent shared states, compatible joint output iff/uniqueness, guarded partial common update and lifts. Conservation limited to two-stock integer cancellation.
11.2 Common realization and viability	**Excluded analytic theorem**. Rational amount/area identities conditional on $0 \leq \xi < 1$ and rational capsule-margin prerequisites. No tanh solution, continuous existence/uniqueness, real capsule theorem, or complete viability formalization.
12.1 Quantitative realization condition	**Excluded**. No Lipschitz-margin transfer formalization.
12.2 Symmetry does not force handedness	**Excluded**. No equivariant ODE uniqueness theorem.

Manuscript statement	Exact Lean scope
C.2 Separation and protected structure	**Protected-current-data kernel only**. Generic editing preserves immutable element/origin maps and every unedited protected current-bond value; arbitrary legal words preserve protected current bonds under explicit step hypotheses. No general deleted-edge path equivalence, valence grammar or chemistry theorem.
C.4 Release inclusion and frame covariance	**Excluded**. No release-set or Euclidean-frame covariance theorem.
C.5 Typed unitary covariance	**Excluded**. No unitary covariance theorem.
C.6 Response stability away from the bridge spectrum	**Excluded**. No general perturbation/operator-norm bound.
C.7 Covariance and nontriviality of the finite response	**Excluded**. No unitary response covariance or causal integral proof.
C.8 Exact and approximate invariant sectors	**Excluded**. No unitary-sector invariance/leakage bound.
C.9 Exact finite encoded channel	**Excluded**. No general completely-positive/trace-preserving encoded-channel theorem.
D.1 Finite split and join construction	**Excluded**. No general finite valence-preserving split/join construction.
E.1 Finite-count bounds	**Integer/rational prerequisites only**. Seven-cell carbon identity for arbitrary integer B,K, conditional inventory guards, and selected rational inequalities; not hybrid existence, event timing, floor-clock analysis or universal attenuation theorem.

5 Checking and recovering the proof

Install the official Lean toolchain identified by the bundled `lean-toolchain` file. Then the raw proof checks with

```
lean --version
lean Article85.lean
```

The version must report Lean 4.24.0. The audit harness checks the exact version, recompiles the source afresh, audits declaration dependencies and exercises selected rejection controls. It does not trust a saved `.olean` file or a historical build log.

5.1 From this TeX alone

Copy only `Article88_Formal_Proof_Overleaf_LaTeX.tex` to an empty writable directory:

```
pdflatex -interaction=nonstopmode -halt-on-error Article88_Formal_Proof_Overleaf_LaTeX.tex
pdflatex -interaction=nonstopmode -halt-on-error Article88_Formal_Proof_Overleaf_LaTeX.tex
python3 extract_article85_formal.py --output-dir article85_formal
python3 article85_formal/audit_lean.py --lean lean --output fresh-lean-audit
```

The TeX build uses ordinary declared packages, needs no shell escape, and does not itself execute Lean. Unicode symbols in the printed listing are typeset through ASCII LaTeX escapes; they are not executable source files. The canonical Unicode `Article85.lean` is recovered from the base64 archive with its exact pinned bytes. No special engine flag is required. Printed Python string literals use equivalent Unicode escape sequences where needed; the archive preserves their original bytes.

5.2 From this digital PDF alone

Save the embedded files, or use Poppler:

```
pdfdetach -saveall Article88_Formal_Proof_Overleaf_LaTeX.pdf
python3 extract_article85_formal.py --output-dir article85_formal
python3 article85_formal/audit_lean.py --lean lean --output fresh-lean-audit
```

The PDF attachments are `article85_formal_payload.b64` and `extract_article85_formal.py`. This pair recovers all audit and toolchain metadata and the canonical raw proof, which suffices for direct checking once the declared toolchain is installed. A screenshot or printout does not preserve attachments.

Recovery validates the pinned archive hash, exact member inventory, member sizes and hashes, and safe nonduplicate relative paths before writing files. It rejects a nonempty output directory. These integrity checks do not replace mathematical review of theorem statements.

6 Evidence and trust boundary

The release was checked with a fresh official pinned toolchain. Every included theorem has its dependency set audited; only Lean's ordinary foundational axioms, when actually reported, are allowed. The per-declaration report is retained in the runtime evidence. The rejection suite includes false conclusions and a compile-valid custom-axiom injection; unexpected acceptance is a failure. Selected negative tests cannot establish completeness of the software audit itself.

Actual isolated TeX-only and PDF-only recovery were followed by fresh compilation and auditing. The companion's source-to-manuscript mapping was also reviewed independently. Successful compilation establishes that the supplied statements follow from their explicit definitions and hypotheses in Lean; semantic correspondence and excluded physical interpretations remain visible obligations, not consequences of a compiler exit status.

6.1 Complete runtime identity

The runtime archive SHA-256 is, split only for display,

```
de6888fa26fb84af3c6698d30f81304c
b477cf3cfdbc289c43409ec2c2bbb8a9
```

The following listings reproduce the complete proof and its audit programs. Printed line wrapping does not alter the recovered source.

7 Complete Lean 4 proof source

Article85.lean

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

/-! Article 85 partial formal certificate. Lean 4.24.0, official Std/core only.
Controlling source: Article 85 revised manuscript dated 11 October 2026.
Read COVERAGE.json for exact general-theorem/finite-kernel distinctions and exclusions.
No full F1-F12 existence, spectral analysis, differential equation, or chemical realization
is asserted by this partial certificate. Every declared theorem is axiom-audited below. -/

set_option maxRecDepth 100000
set_option maxHeartbeats 0

/- ===== Recognition.lean: start ===== -/

/-!
Article 85: coordinate recognition, proved for arbitrary connected symmetric
labelled graphs (hence for the finite directional graphs of the manuscript).
No finiteness, molecular, spectral, chemical, or physical premise is hidden.
The direction restriction is used only for the induced-neighbor conclusion.
-/
namespace Article85

abbrev Vec3 := Fin 3 → Int

structure LabelledGraph (V : Type) where
  edge : V → V → Prop
  symmetric : ∀ {x y}, edge x y → edge y x
  step : V → V → Vec3
  antisymmetric : ∀ {x y}, edge x y → ∀ i, step y x i = -step x y i

inductive Walk {V : Type} (G : LabelledGraph V) : V → V → Type where
  | nil (x) : Walk G x x
  | cons {x y z} : G.edge x y → Walk G y z → Walk G x z

namespace Walk
variable {V : Type} {G : LabelledGraph V}

def append {x y z} : Walk G x y → Walk G y z → Walk G x z
  | .nil _, q => q
  | .cons h p, q => .cons h (append p q)

def reverse {x y} : Walk G x y → Walk G y x
  | .nil x => .nil x
  | .cons h p => append (reverse p) (.cons (G.symmetric h) (.nil _))

def integral {x y} : Walk G x y → Vec3
  | .nil _ => fun _ => 0
  | .cons (x := x) (y := y) _ p => fun i => G.step x y i + integral p i

theorem integral_append {x y z} (p : Walk G x y) (q : Walk G y z) (i) :
    (append p q).integral i = p.integral i + q.integral i := by
  induction p with
  | nil => simp [append, integral]
  | cons h p ih => simp [append, integral, ih, Int.add_assoc]
```

```
theorem integral_reverse {x y} (p : Walk G x y) (i) :
    p.reverse.integral i = -p.integral i := by
  induction p with
  | nil => simp [reverse, integral]
  | cons h p ih =>
    simp only [reverse, integral_append, integral]
    have hs := G.antisymmetric h i
    omega
end Walk

variable {V : Type} (G : LabelledGraph V)

def Connected : Prop := ∀ x y, Nonempty (Walk G x y)
def Coordinate (a : V → Vec3) : Prop :=
  ∀ x y, G.edge x y → ∀ i, a y i - a x i = G.step x y i

def ZeroPeriods : Prop := ∀ x, ∀ p : Walk G x x, ∀ i, p.integral i = 0

def WalkSeparates : Prop :=
  ∀ x y, ∀ p : Walk G x y, (∀ i, p.integral i = 0) → x = y

theorem integral_telescope (a : V → Vec3) (ha : Coordinate G a)
    {x y} (p : Walk G x y) (i) :
    p.integral i = a y i - a x i := by
  induction p with
  | nil => simp [Walk.integral]
  | cons h p ih =>
    simp only [Walk.integral]
    have hs := ha _ _ h i
    omega

theorem coordinate_zero_periods (a : V → Vec3) (ha : Coordinate G a) :
    ZeroPeriods G := by
  intro x p i
  rw [integral_telescope G a ha p i]
  omega

noncomputable def rootCoordinate (hconn : Connected G) (root : V) : V → Vec3 :=
  fun x => (Classical.choice (hconn root x)).integral

theorem zero_periods_coordinate (hconn : Connected G) (root : V)
    (hz : ZeroPeriods G) : Coordinate G (rootCoordinate G hconn root) := by
  intro x y hxy i
  let px := Classical.choice (hconn root x)
  let py := Classical.choice (hconn root y)
  let e : Walk G x y := .cons hxy (.nil y)
  have hzloop := hz root ((px.append e).append py.reverse) i
  rw [Walk.integral_append, Walk.integral_append, Walk.integral_reverse] at hzloop
  change px.integral i + (G.step x y i + 0) + -py.integral i = 0 at hzloop
  change py.integral i - px.integral i = G.step x y i
  omega

theorem recognition_existence (hconn : Connected G) (root : V) :
    (∃ a, Coordinate G a) ↔ ZeroPeriods G := by
  constructor
  · rintro ⟨a, ha⟩
    exact coordinate_zero_periods G a ha
  · intro hz
```

```
    exact ⟨rootCoordinate G hconn root, zero_periods_coordinate G hconn root hz⟩

theorem coordinate_unique_translation (hconn : Connected G) (root : V)
    (a b : V → Vec3) (ha : Coordinate G a) (hb : Coordinate G b) :
    ∀ x i, b x i - a x i = b root i - a root i := by
  intro x i
  let p := Classical.choice (hconn root x)
  have h1 := integral_telescope G a ha p i
  have h2 := integral_telescope G b hb p i
  omega

theorem coordinate_injective_iff (hconn : Connected G)
    (a : V → Vec3) (ha : Coordinate G a) :
    Function.Injective a ↔ WalkSeparates G := by
  constructor
  · intro hinj x y p hp
    apply hinj
    funext i
    have hi := integral_telescope G a ha p i
    have hz := hp i
    omega
  · intro hsep x y heq
    let p := Classical.choice (hconn x y)
    apply hsep x y p
    intro i
    rw [integral_telescope G a ha p i, heq]
    omega

/-- Six signed unit vectors, defined without wraparound. -/
def IsUnitStep (v : Vec3) : Prop :=
  ∃ axis : Fin 3, (v axis = 1 ∨ v axis = -1) ∧
    ∀ j, j ≠ axis → v j = 0

def NeighborComplete (a : V → Vec3) : Prop :=
  ∀ x y, IsUnitStep (fun i => a y i - a x i) → G.edge x y

def InducedIncidence (a : V → Vec3) : Prop :=
  ∀ x y, G.edge x y ↔ IsUnitStep (fun i => a y i - a x i)

theorem induced_incidence_iff (a : V → Vec3) (ha : Coordinate G a)
    (hdir : ∀ x y, G.edge x y → IsUnitStep (G.step x y)) :
    InducedIncidence G a ↔ NeighborComplete G a := by
  constructor
  · intro hi x y hn
    exact (hi x y).mpr hn
  · intro hc x y
    constructor
    · intro he
      have hv : (fun i => a y i - a x i) = G.step x y := by
        funext i
        exact ha x y he i
      rw [hv]
      exact hdir x y he
    · exact hc x y

/-- All three manuscript recognition conditions, including separation and
incidence completeness, are equivalent to an injective induced realization. -/
theorem recognition_full (hconn : Connected G) (root : V)
    (hdir : ∀ x y, G.edge x y → IsUnitStep (G.step x y)) :
```

```
    (∃ a, Coordinate G a ∧ Function.Injective a ∧ InducedIncidence G a) ↔
    (ZeroPeriods G ∧ WalkSeparates G ∧
      NeighborComplete G (rootCoordinate G hconn root)) := by
  constructor
· rintro ⟨a, ha, hinj, hind⟩
  have hz := coordinate_zero_periods G a ha
  refine ⟨hz, (coordinate_injective_iff G hconn a ha).mp hinj, ?_⟩
  intro x y hn
  have hr := zero_periods_coordinate G hconn root hz
  have hx := coordinate_unique_translation G hconn root a
    (rootCoordinate G hconn root) ha hr x
  have hy := coordinate_unique_translation G hconn root a
    (rootCoordinate G hconn root) ha hr y
  have heq : (fun i => rootCoordinate G hconn root y i -
      rootCoordinate G hconn root x i) = (fun i => a y i - a x i) := by
    funext i
    have hxi := hx i
    have hyi := hy i
    omega
  rw [heq] at hn
  exact (hind x y).mpr hn
· rintro ⟨hz, hsep, hc⟩
  have ha := zero_periods_coordinate G hconn root hz
  exact ⟨rootCoordinate G hconn root, ha,
    (coordinate_injective_iff G hconn _ ha).mpr hsep,
    (induced_incidence_iff G _ ha hdir).mpr hc⟩

/-- Translation composition requires the intermediate point; no equality of
maximal domains is asserted. -/
def partialTranslate (region : Vec3 → Prop) [DecidablePred region]
    (m : Vec3) (k : Vec3) : Option Vec3 :=
  if region k ∧ region (fun i => k i + m i) then
    some (fun i => k i + m i) else none

theorem translation_compose_on_domain (region : Vec3 → Prop) [DecidablePred region]
    (k m n : Vec3) (hk : region k) (hn : region (fun i => k i + n i))
    (hm : region (fun i => k i + n i + m i)) :
    (partialTranslate region n k).bind (partialTranslate region m) =
      partialTranslate region (fun i => n i + m i) k := by
  have heq : (fun i => k i + (n i + m i)) = (fun i => k i + n i + m i) := by
    funext i
    omega
  simp [partialTranslate, hk, hn, hm, heq]

end Article85
/- ===== Recognition.lean: end ===== -/

/- ===== Systems.lean: start ===== -/

/-!
Article 85: partial dynamics, finite words, shared-state compatibility and
rational terminal laws. These theorems do not assert a chemical realization.
Option.none is an undefined operation, not a silently unchanged state.
-/

namespace Article85

universe u v w r
```

```
/-- `none` records a failed guard; `some` records its complete new state. -/
abbrev PartialActions (U : Type u) (X : Type v) := U → X → Option X

def executeWord {U : Type u} {X : Type v} (f : PartialActions U X) :
    List U → X → Option X
  | [], x => some x
  | a :: word, x => (f a x).bind (executeWord f word)

/-- Equality includes both output and domain, including undefined inputs. -/
def ExactIntertwining {U : Type u} {X : Type v} {Y : Type w}
    (f : PartialActions U X) (g : PartialActions U Y) (p : X → Y) : Prop :=
  ∀ a x, (f a x).map p = g a (p x)

/-- One-step exact intertwining suffices for every finite word. -/
theorem intertwining_all_words {U : Type u} {X : Type v} {Y : Type w}
    (f : PartialActions U X) (g : PartialActions U Y) (p : X → Y)
    (h : ExactIntertwining f g p) (word : List U) (x : X) :
    (executeWord f word x).map p = executeWord g word (p x) := by
  induction word generalizing x with
  | nil => rfl
  | cons a word ih =>
    simp only [executeWord]
    rw [← h a x]
    cases hf : f a x with
    | none => rfl
    | some x' => simpa using ih x'

theorem intertwining_identity {U : Type u} {X : Type v}
    (f : PartialActions U X) : ExactIntertwining f f id := by
  intro a x
  change (f a x).map id = f a x
  cases f a x <;> rfl

theorem intertwining_compose {U : Type u} {X : Type v} {Y : Type w}
    {Z : Type r} (f : PartialActions U X) (g : PartialActions U Y)
    (h : PartialActions U Z) (p : X → Y) (q : Y → Z)
    (hp : ExactIntertwining f g p) (hq : ExactIntertwining g h q) :
    ExactIntertwining f h (q ∘ p) := by
  intro a x
  change (f a x).map (q ∘ p) = h a (q (p x))
  rw [← hq a (p x), ← hp a x]
  cases f a x <;> rfl

structure EncodedRealization {U : Type u} {X : Type v} {Y : Type w}
    (f : PartialActions U X) (g : PartialActions U Y) where
  encode : X → Y
  injective : Function.Injective encode
  step : ExactIntertwining f g encode

structure ExactQuotient {U : Type u} {X : Type v} {Y : Type w}
    (f : PartialActions U X) (g : PartialActions U Y) where
  project : Y → X
  surjective : Function.Surjective project
  step : ExactIntertwining g f project

def EncodedRealization.comp {U : Type u} {X : Type v} {Y : Type w} {Z : Type r}
    {f : PartialActions U X} {g : PartialActions U Y} {h : PartialActions U Z}
    (e : EncodedRealization f g) (k : EncodedRealization g h) :
    EncodedRealization f h where
```

```
    encode := k.encode ∘ e.encode
    injective := fun _ _ he => e.injective (k.injective he)
    step := intertwining_compose f g h e.encode k.encode e.step k.step

def ExactQuotient.comp {U : Type u} {X : Type v} {Y : Type w} {Z : Type r}
    {f : PartialActions U X} {g : PartialActions U Y} {h : PartialActions U Z}
    (p : ExactQuotient f g) (q : ExactQuotient g h) : ExactQuotient f h where
  project := p.project ∘ q.project
  surjective := by
    intro x
    obtain ⟨y, hy⟩ := p.surjective x
    obtain ⟨z, hz⟩ := q.surjective y
    exact ⟨z, by simp [hz, hy]⟩
  step := intertwining_compose h g f q.project p.project q.step p.step

def LegalWord {U : Type u} {X : Type v} (f : PartialActions U X)
    (word : List U) (x : X) : Prop := ∃ y, executeWord f word x = some y

/-- Both encodings and quotient projections preserve and reflect legality. -/
theorem intertwining_legality_iff {U : Type u} {X : Type v} {Y : Type w}
    (f : PartialActions U X) (g : PartialActions U Y) (p : X → Y)
    (h : ExactIntertwining f g p) (word : List U) (x : X) :
    LegalWord f word x ↔ LegalWord g word (p x) := by
  have hw := intertwining_all_words f g p h word x
  cases he : executeWord f word x with
  | none => simp [LegalWord, he, ← hw]
  | some y => simp [LegalWord, he, ← hw]

/-- Every legal source execution lifts from every state above its initial state. -/
theorem quotient_lifts_word {U : Type u} {X : Type v} {Y : Type w}
    {f : PartialActions U X} {g : PartialActions U Y}
    (p : ExactQuotient f g) (word : List U) (y : Y) (x' : X)
    (hx : executeWord f word (p.project y) = some x') :
    ∃ y', executeWord g word y = some y' ∧ p.project y' = x' := by
  have hw := intertwining_all_words g f p.project p.step word y
  rw [hx] at hw
  cases he : executeWord g word y with
  | none => simp [he] at hw
  | some y' =>
    have hp : p.project y' = x' := by simpa [he] using hw
    exact ⟨y', rfl, hp⟩

/-- The whole encoded image, not merely the decoded value, is invariant. -/
theorem encoded_word_output {U : Type u} {X : Type v} {Y : Type w}
    {f : PartialActions U X} {g : PartialActions U Y}
    (e : EncodedRealization f g) (word : List U) (x x' : X)
    (hx : executeWord f word x = some x') :
    executeWord g word (e.encode x) = some (e.encode x') := by
  rw [← intertwining_all_words f g e.encode e.step word x, hx]
  rfl

/-- Includes the initial observation and every successful intermediate one. -/
def observedTrace {U : Type u} {X : Type v} {O : Type w}
    (f : PartialActions U X) (obs : X → O) : List U → X → Option (List O)
  | [], x => some [obs x]
  | a :: word, x => (f a x).bind fun x' =>
      (observedTrace f obs word x').map (fun rest => obs x :: rest)

theorem intertwining_observed_trace {U : Type u} {X : Type v} {Y : Type w}
```

```
    {O : Type r} (f : PartialActions U X) (g : PartialActions U Y)
    (p : X → Y) (obsX : X → O) (obsY : Y → O)
    (h : ExactIntertwining f g p) (ho : ∀ x, obsY (p x) = obsX x)
    (word : List U) (x : X) :
    observedTrace f obsX word x = observedTrace g obsY word (p x) := by
  induction word generalizing x with
  | nil => simp [observedTrace, ho]
  | cons a word ih =>
    simp only [observedTrace]
    rw [← h a x]
    cases hf : f a x with
    | none => rfl
    | some x' => simpa [ho] using congrArg (fun rest => rest.map (fun r => obsX x :: r)) (ih
     x')

/-- Preserving a predicate at each defined step preserves it on legal words. -/
theorem word_preserves_predicate {U : Type u} {X : Type v}
    (f : PartialActions U X) (P : X → Prop)
    (h : ∀ a x y, P x → f a x = some y → P y)
    (word : List U) (x y : X) (hx : P x)
    (hw : executeWord f word x = some y) : P y := by
  induction word generalizing x y with
  | nil =>
    have he : x = y := Option.some.inj hw
    exact he ▸ hx
  | cons a word ih =>
    cases hs : f a x with
    | none => simp [executeWord, hs] at hw
    | some x' =>
      have hx' := h a x x' hx hs
      exact ih x' y hx' (by simpa [executeWord, hs] using hw)

theorem word_conserves_quantity {U : Type u} {X : Type v} {C : Type w}
    (f : PartialActions U X) (c : X → C)
    (h : ∀ a x y, f a x = some y → c y = c x)
    (word : List U) (x y : X) (hw : executeWord f word x = some y) :
    c y = c x := by
  apply word_preserves_predicate f (fun z => c z = c x) _ word x y rfl hw
  intro a z z' hz hs
  exact (h a z z' hs).trans hz

/-- A true product-coordinate independence condition includes guards. -/
def updateLeft {X : Type u} {Y : Type v} (f : X → Option X) (s : X × Y) : Option (X × Y)
    :=
  (f s.1).map (fun x => (x, s.2))

def updateRight {X : Type u} {Y : Type v} (g : Y → Option Y) (s : X × Y) : Option (X × Y)
    :=
  (g s.2).map (fun y => (s.1, y))

theorem independent_guarded_updates_commute {X : Type u} {Y : Type v}
    (f : X → Option X) (g : Y → Option Y) (s : X × Y) :
    (updateLeft f s).bind (updateRight g) =
      (updateRight g s).bind (updateLeft f) := by
  cases s with
  | mk x y =>
    cases hf : f x <;> cases hg : g y <;> simp [updateLeft, updateRight, hf, hg]

/-- A negative control: disjoint writes do not imply noninterference. -/
```

```
def enableFlag (s : Bool × Bool) : Option (Bool × Bool) := some (true, s.2)
def guardedRegister (s : Bool × Bool) : Option (Bool × Bool) :=
  if s.1 then some (s.1, !s.2) else none

theorem disjoint_writes_can_change_domain :
    (enableFlag (false, false)).bind guardedRegister ≠
      (guardedRegister (false, false)).bind enableFlag := by
  decide +kernel

/-- A negative control for replacing image invariance by one-step decoding alone. -/
def weakEncode (b : Bool) : Bool × Bool := (b, false)
def weakDecode (s : Bool × Bool) : Bool := s.1
def weakTarget (s : Bool × Bool) : Bool × Bool :=
  (if s.2 then s.1 else !s.1, true)

theorem one_step_decoding_is_not_iteration :
    (∀ b, weakDecode (weakTarget (weakEncode b)) = !b) ∧
    weakDecode (weakTarget (weakTarget (weakEncode false))) ≠ !(!false) := by
  decide +kernel

/-- Indexed shared-state equality, allowing different subsystem types. -/
def SharedCompatible {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) (x : (i : I) → X i) : Prop :=
  ∀ i j, q i (x i) = q j (x j)

abbrev SharedState {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) := {x : (i : I) → X i // SharedCompatible q x}

/-- On a proposed common domain, an update exists exactly when readings agree. -/
theorem shared_update_exists_iff {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) (y : (i : I) → X i) :
    (∃ out : SharedState q, ∀ i, out.val i = y i) ↔ SharedCompatible q y := by
  constructor
  · intro ⟨out, hout⟩ i j
    rw [← hout i, ← hout j]
    exact out.property i j
  · intro h
    exact ⟨⟨y, h⟩, fun _ => rfl⟩

theorem shared_update_unique {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) (a b : SharedState q)
    (h : ∀ i, a.val i = b.val i) : a = b := by
  apply Subtype.ext
  exact funext h

/-- A common update of readings is sufficient, even for heterogeneous subsystems. -/
theorem shared_common_update {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) (T : (i : I) → X i → X i) (w : Z → Z)
    (h : ∀ i x, q i (T i x) = w (q i x)) (x : SharedState q) :
    SharedCompatible q (fun i => T i (x.val i)) := by
  intro i j
  rw [h i, h j, x.property i j]

/-- Tagged lifts of one common state give a nonempty shared fiber at that state. -/
theorem shared_lifts_compatible {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) (lift : (i : I) → Z → X i)
    (h : ∀ i z, q i (lift i z) = z) (z : Z) :
    SharedCompatible q (fun i => lift i z) := by
  intro i j
```

```
  rw [h i, h j]

/-- All subsystem guards and agreement of their returned shared readings are required. -/
theorem shared_partial_update {I : Type u} {X : I → Type v} {Z : Type w}
    (q : (i : I) → X i → Z) (T : (i : I) → X i → Option (X i))
    (w : Z → Z) (x : SharedState q)
    (legal : ∀ i, ∃ y, T i (x.val i) = some y)
    (related : ∀ i y, T i (x.val i) = some y → q i y = w (q i (x.val i))) :
    ∃ out : SharedState q,
      (∀ i, T i (x.val i) = some (out.val i)) ∧
      ∀ other : SharedState q,
        (∀ i, T i (x.val i) = some (other.val i)) → other = out := by
  classical
  let y := fun i => Classical.choose (legal i)
  have hy : ∀ i, T i (x.val i) = some (y i) := fun i => Classical.choose_spec (legal i)
  have hc : SharedCompatible q y := by
    intro i j
    rw [related i (y i) (hy i), related j (y j) (hy j), x.property i j]
  refine ⟨⟨y, hc⟩, hy, ?_⟩
  intro other ho
  apply shared_update_unique q
  intro i
  exact Option.some.inj ((ho i).symm.trans (hy i))

/-- Interface flux cancellation with neither stock duplicated. -/
theorem two_stock_flux_cancels (a b a' b' flux : Int)
    (ha : a' = a - flux) (hb : b' = b + flux) :
    a' + b' = a + b := by omega

/-- Rational scalar Schur complement; invertibility is an explicit premise. -/
def scalarSchur (a b c d : Rat) : Rat := a - b * (1 / d) * c

theorem scalar_schur_elimination (a b c d x y : Rat) (hd : d ≠ 0)
    (hi : c * x + d * y = 0) :
    a * x + b * y = scalarSchur a b c d * x := by
  have hdInv : d * (1 / d) = 1 := by grind
  unfold scalarSchur
  grind

/-- Path midpoint elimination for every terminal potential and conductance. -/
theorem path_terminal_current (g vMinus vPlus : Rat) :
    (2 * g) * (vPlus - (vMinus + vPlus) / 2) = g * (vPlus - vMinus) := by
  grind

/-- The two cycle branches have the same combined law as the path. -/
theorem cycle_terminal_current (g vMinus vPlus : Rat) :
    g * (vPlus - (vMinus + vPlus) / 2) +
      g * (vPlus - (vMinus + vPlus) / 2) = g * (vPlus - vMinus) := by
  grind

theorem midpoint_kirchhoff (g vMinus vPlus : Rat) :
    g * ((vMinus + vPlus) / 2 - vMinus) +
      g * ((vMinus + vPlus) / 2 - vPlus) = 0 := by
  grind

theorem midpoint_unique (g vMinus vPlus vInterior : Rat) (hg : g ≠ 0)
    (hK : g * (vInterior - vMinus) + g * (vInterior - vPlus) = 0) :
    vInterior = (vMinus + vPlus) / 2 := by
  grind
```

```
/-- Reduced equations include arbitrary common context currents at every node. -/
def ReducedContextSolution {I : Type u}
    (moduleLaw contextCurrent drive : (I → Rat) → I → Rat)
    (v : I → Rat) : Prop :=
  ∀ i, moduleLaw v i + contextCurrent v i = drive v i

/-- Equal boundary laws give equal solutions under a unique common Dirichlet problem. -/
theorem contextual_terminal_replacement {I : Type u} {O : Type v}
    (lawA lawB ctx drive : (I → Rat) → I → Rat)
    (external : (I → Rat) → O)
    (sameLaw : ∀ v i, lawA v i = lawB v i)
    (unique : ∀ v w, ReducedContextSolution lawA ctx drive v →
      ReducedContextSolution lawA ctx drive w → v = w)
    (vA vB : I → Rat)
    (hA : ReducedContextSolution lawA ctx drive vA)
    (hB : ReducedContextSolution lawB ctx drive vB) :
    external vA = external vB := by
  apply congrArg external
  apply unique vA vB hA
  intro i
  rw [sameLaw]
  exact hB i

end Article85


/- ===== Systems.lean: end ===== -/

/- ===== Genetics.lean: start ===== -/

/-! Article 85 fixed-AAA addressing and typed component-control certificate.
No operation creates a resident species. The carrier type has exactly AAA;
its mutable six-bit address is a different field. State initialization and
operation selection have different Lean types. -/
namespace Article85

structure Word6 where
  b1 : Bool
  b2 : Bool
  b3 : Bool
  b4 : Bool
  b5 : Bool
  b6 : Bool
  deriving DecidableEq, Repr

def complement (w : Word6) : Word6 :=
  ⟨!w.b1, !w.b2, !w.b3, !w.b4, !w.b5, !w.b6⟩
def reversal (w : Word6) : Word6 :=
  ⟨w.b6, w.b5, w.b4, w.b3, w.b2, w.b1⟩

theorem complement_involution (w : Word6) : complement (complement w) = w := by
  cases w
  simp [complement]
theorem reversal_involution (w : Word6) : reversal (reversal w) = w := by
  rfl
theorem complement_reversal_commute (w : Word6) :
    complement (reversal w) = reversal (complement w) := by rfl

```

```
inductive Resident where
  | AAA
  deriving DecidableEq, Repr

structure Address where
  word : Word6
  complementTag : Bool
  directionTag : Bool
  resident : Resident
  deriving DecidableEq, Repr

def toggleF (p : Address) : Address :=
  { p with word := complement p.word, complementTag := !p.complementTag }
def toggleR (p : Address) : Address :=
  { p with word := reversal p.word, directionTag := !p.directionTag }
def normalWord (p : Address) : Word6 :=
  let w := if p.directionTag then reversal p.word else p.word
  if p.complementTag then complement w else w

def canonical (w : Word6) : Address := ⟨w, false, false, .AAA⟩
def refAction (a d : Bool) (p : Address) : Address :=
  let q := if d then toggleR p else p
  if a then toggleF q else q

def SameOrbit (p q : Address) : Prop := ∃ a d, refAction a d p = q

theorem toggleF_involution (p : Address) : toggleF (toggleF p) = p := by
  cases p
  simp [toggleF, complement_involution]
theorem toggleR_involution (p : Address) : toggleR (toggleR p) = p := by
  cases p
  simp [toggleR, reversal_involution]
theorem reference_actions_commute (p : Address) :
    toggleF (toggleR p) = toggleR (toggleF p) := by rfl

theorem normalWord_F (p : Address) : normalWord (toggleF p) = normalWord p := by
  rcases p with ⟨w, a, d, t⟩
  cases a <;> cases d <;>
    simp [normalWord, toggleF, complement_involution, ← complement_reversal_commute]
theorem normalWord_R (p : Address) : normalWord (toggleR p) = normalWord p := by
  rcases p with ⟨w, a, d, t⟩
  cases a <;> cases d <;> simp [normalWord, toggleR, reversal_involution]
theorem normalWord_action (a d : Bool) (p : Address) :
    normalWord (refAction a d p) = normalWord p := by
  cases a <;> cases d <;> simp [refAction, normalWord_F, normalWord_R]
theorem action_preserves_resident (a d : Bool) (p : Address) :
    (refAction a d p).resident = p.resident := by
  cases a <;> cases d <;> rfl

theorem action_involution (a d : Bool) (p : Address) :
    refAction a d (refAction a d p) = p := by
  cases a <;> cases d <;>
    simp [refAction, toggleF_involution, toggleR_involution,
      ← reference_actions_commute]

theorem reaches_normal_form (p : Address) :
    refAction p.complementTag p.directionTag p = canonical (normalWord p) := by
  rcases p with ⟨w, a, d, t⟩
  cases t
```

```
  cases a <;> cases d <;> rfl

theorem zero_tag_representative_unique (p : Address)
    (ha : p.complementTag = false) (hd : p.directionTag = false) :
    p = canonical (normalWord p) := by
  rcases p with ⟨w, a, d, t⟩
  cases t
  simp_all [canonical, normalWord]

theorem refAction_compose (a d b e : Bool) (p : Address) :
    refAction a d (refAction b e p) = refAction (a ^^ b) (d ^^ e) p := by
  cases a <;> cases d <;> cases b <;> cases e <;>
    simp [refAction, toggleF_involution, toggleR_involution,
      ← reference_actions_commute]

theorem orbit_iff_normal (p q : Address) : SameOrbit p q ↔ normalWord p = normalWord q :=
    by
  constructor
  · rintro ⟨a, d, rfl⟩
    exact (normalWord_action a d p).symm
  · intro h
    have hp := reaches_normal_form p
    have hq := reaches_normal_form q
    rw [h] at hp
    have hc : refAction q.complementTag q.directionTag
      (refAction p.complementTag p.directionTag p) = q := by
      rw [hp, ← hq, action_involution]
    rw [refAction_compose] at hc
    exact ⟨_, _, hc⟩

/-- A codon is three bases. No biological output map is inferred here. -/
inductive Base where
  | A | C | G | U
  deriving DecidableEq, Repr

def pairBase : Bool → Bool → Base
  | false, false => .A
  | false, true => .C
  | true, false => .G
  | true, true => .U

def baseBits : Base → Bool × Bool
  | .A => (false, false)
  | .C => (false, true)
  | .G => (true, false)
  | .U => (true, true)

abbrev Codon := Base × Base × Base

def encodeCodon (w : Word6) : Codon :=
  (pairBase w.b1 w.b2, pairBase w.b3 w.b4, pairBase w.b5 w.b6)
def decodeCodon (c : Codon) : Word6 :=
  ⟨(baseBits c.1).1, (baseBits c.1).2,
   (baseBits c.2.1).1, (baseBits c.2.1).2,
   (baseBits c.2.2).1, (baseBits c.2.2).2⟩
def swapCG : Base → Base
  | .C => .G
  | .G => .C
  | b => b
```

```
def transportedReverse (c : Codon) : Codon :=
  (swapCG c.2.2, swapCG c.2.1, swapCG c.1)

theorem pairBase_bits (a b : Bool) : baseBits (pairBase a b) = (a,b) := by
  cases a <;> cases b <;> rfl
theorem baseBits_base (b : Base) : pairBase (baseBits b).1 (baseBits b).2 = b := by
  cases b <;> rfl
theorem codon_roundtrip (w : Word6) : decodeCodon (encodeCodon w) = w := by
  cases w
  simp [decodeCodon, encodeCodon, pairBase_bits]
theorem word_roundtrip (c : Codon) : encodeCodon (decodeCodon c) = c := by
  rcases c with ⟨a,b,c⟩
  simp [decodeCodon, encodeCodon, baseBits_base]
theorem pairBase_reversal (a b : Bool) : pairBase b a = swapCG (pairBase a b) := by
  cases a <;> cases b <;> rfl
theorem transported_reversal (w : Word6) :
    encodeCodon (reversal w) = transportedReverse (encodeCodon w) := by
  simp only [encodeCodon, reversal, transportedReverse]
  rw [pairBase_reversal w.b5 w.b6, pairBase_reversal w.b3 w.b4, pairBase_reversal w.b1 w.b2]

abbrev Register := Bool × Bool

def registerUpdate (x : Register) : Register := (x.1, x.2 ^^ x.1)
def GCG : Word6 := ⟨true,false,false,true,true,false⟩
def GCU : Word6 := ⟨true,false,false,true,true,true⟩
def selected (w : Word6) : Prop := w = GCG ∨ w = GCU
instance (w : Word6) : Decidable (selected w) := inferInstanceAs (Decidable (w = GCG ∨ w =
    GCU))

/-- State-valued initialization on exactly the selected address domain. -/
def initState (w : Word6) : Option Register :=
  if selected w then some (!w.b6, true) else none

inductive ControlInput where
  | update
  | prep (x : Register)
  deriving DecidableEq, Repr

/-- Operation-valued selector, not identified with the state initialization. -/
def selectInput (w : Word6) : Option ControlInput := (initState w).map ControlInput.prep

def runInput : ControlInput → Register → Register
  | .update, x => registerUpdate x
  | .prep x, _ => x

def dispatch (p : Address) : Option ControlInput := selectInput (normalWord p)

def dispatchAction (p : Address) (x : Register) : Option Register :=
  (dispatch p).map (fun u => runInput u x)

theorem selected_initializations : initState GCG = some (true,true) ∧
    initState GCU = some (false,true) := by decide

theorem selected_codons : encodeCodon GCG = (.G,.C,.G) ∧
    encodeCodon GCU = (.G,.C,.U) := by decide

theorem initState_domain (w : Word6) : (initState w).isSome = true ↔ selected w := by
  simp [initState]
```

```
theorem dispatch_reference_invariant (a d : Bool) (p : Address) :
    dispatch (refAction a d p) = dispatch p := by
  simp [dispatch, normalWord_action]

theorem preparation_is_initialization (p : Address) (x : Register) :
    dispatchAction p x = initState (normalWord p) := by
  simp only [dispatchAction, dispatch, selectInput]
  cases h : initState (normalWord p) <;> rfl

theorem register_update_nonidentity : registerUpdate (true,false) = (true,true) ∧
    registerUpdate (true,false) ≠ (true,false) := by decide

theorem register_update_involution (x : Register) : registerUpdate (registerUpdate x) = x :=
     by
  rcases x with ⟨m,r⟩
  cases m <;> cases r <;> rfl

theorem initialization_distinct_from_update :
    dispatchAction (canonical GCG) (false,false) = some (true,true) ∧
    registerUpdate (false,false) = (false,false) := by decide

/-- Resident populations are supplied state, not manufactured by addressing.
An arbitrary existing population indexed by K is preserved pointwise. -/
structure GeneticState (K : Type) where
  population : K → Resident
  address : Address
  register : Register

def operateGenetic {K : Type} (u : ControlInput) (s : GeneticState K) : GeneticState K :=
  { s with register := runInput u s.register }

theorem operation_preserves_population {K : Type} (u : ControlInput) (s : GeneticState K) :
    (operateGenetic u s).population = s.population := by rfl

theorem operation_preserves_address {K : Type} (u : ControlInput) (s : GeneticState K) :
    (operateGenetic u s).address = s.address := by rfl

/-- Exhaustive six-bit enumeration, with no sample-based quotient count. -/
def allWords : List Word6 := [false,true].flatMap fun a =>
  [false,true].flatMap fun b => [false,true].flatMap fun c =>
  [false,true].flatMap fun d => [false,true].flatMap fun e =>
  [false,true].map fun f => ⟨a,b,c,d,e,f⟩

def wordNumber (w : Word6) : Nat :=
  (if w.b1 then 32 else 0)+(if w.b2 then 16 else 0)+
  (if w.b3 then 8 else 0)+(if w.b4 then 4 else 0)+
  (if w.b5 then 2 else 0)+(if w.b6 then 1 else 0)

def reversalClass (w : Word6) : Nat := min (wordNumber w) (wordNumber (reversal w))
def untaggedClass (w : Word6) : Nat :=
  min (reversalClass w) (reversalClass (complement w))
def allAddresses : List Address := allWords.flatMap fun w =>
  [false,true].flatMap fun a => [false,true].map fun d => ⟨w,a,d,.AAA⟩

theorem allWords_complete (w : Word6) : w ∈ allWords := by
  rcases w with ⟨a,b,c,d,e,f⟩
  cases a <;> cases b <;> cases c <;> cases d <;> cases e <;> cases f <;> decide
```

```
theorem address_enumeration_counts : allWords.length = 64 ∧ allWords.Nodup ∧
    allAddresses.length = 256 ∧ allAddresses.Nodup := by decide

theorem referenced_normal_form_count :
    (allAddresses.map normalWord).eraseDups.length = 64 := by decide

theorem coarser_quotient_counts :
    (allWords.map reversalClass).eraseDups.length = 36 ∧
    (allWords.map untaggedClass).eraseDups.length = 20 := by decide


end Article85
/- ===== Genetics.lean: end ===== -/

/- ===== Components.lean: start ===== -/

/-! Article 85 finite component-error kernels. Three- and four-site blocks,
full radius-one domains, and exact E/u/D action identities. The code is not
an assertion that a path or cycle is a complete molecule. -/
namespace Article85
namespace Component

abbrev C3 := Bool × Bool × Bool
abbrev C4 := Bool × Bool × Bool × Bool
abbrev Reg := Bool × Bool

def bit (b : Bool) : Nat := if b then 1 else 0
def diff (a b : Bool) : Nat := if a = b then 0 else 1

def E3 (b : Bool) : C3 := (b,b,b)
def E4 (b : Bool) : C4 := (b,b,!b,!b)
def H3 (a b : C3) : Nat := diff a.1 b.1 + diff a.2.1 b.2.1 + diff a.2.2 b.2.2
def H4 (a b : C4) : Nat := diff a.1 b.1 + diff a.2.1 b.2.1 +
  diff a.2.2.1 b.2.2.1 + diff a.2.2.2 b.2.2.2

def ones3 (w : C3) := bit w.1 + bit w.2.1 + bit w.2.2
def correctedOnes4 (w : C4) := bit w.1 + bit w.2.1 + bit (!w.2.2.1) + bit (!w.2.2.2)
def D3 (w : C3) : Bool := 2 ≤ ones3 w
def D4 (w : C4) : Option Bool :=
  if correctedOnes4 w = 2 then none else some (2 < correctedOnes4 w)

theorem E3_distance : H3 (E3 false) (E3 true) = 3 := by decide
theorem E4_distance : H4 (E4 false) (E4 true) = 4 := by decide

theorem one_error_corrected3 (b : Bool) (w : C3) (h : H3 w (E3 b) ≤ 1) : D3 w = b := by
  rcases w with ⟨a,c,d⟩
  cases b <;> cases a <;> cases c <;> cases d <;> revert h <;> decide

theorem one_error_corrected4 (b : Bool) (w : C4) (h : H4 w (E4 b) ≤ 1) : D4 w = some b := by
  rcases w with ⟨a,c,d,e⟩
  cases b <;> cases a <;> cases c <;> cases d <;> cases e <;> revert h <;> decide

theorem roundtrip3 (b : Bool) : D3 (E3 b) = b := by cases b <;> rfl
theorem roundtrip4 (b : Bool) : D4 (E4 b) = some b := by cases b <;> rfl

theorem balls_disjoint3 (w : C3) (h0 : H3 w (E3 false) ≤ 1)
    (h1 : H3 w (E3 true) ≤ 1) : False := by
  have h := one_error_corrected3 false w h0
  have h' := one_error_corrected3 true w h1
```

```
  simp_all

theorem balls_disjoint4 (w : C4) (h0 : H4 w (E4 false) ≤ 1)
    (h1 : H4 w (E4 true) ≤ 1) : False := by
  have h := one_error_corrected4 false w h0
  have h' := one_error_corrected4 true w h1
  simp_all

/-- Generic exact action kernel, specialized below to the finite codes. -/
def update (x : Reg) : Reg := (x.1, x.2 ^^ x.1)
def enc3 (x : Reg) : C3 × C3 := (E3 x.1, E3 x.2)
def dec3 (w : C3 × C3) : Reg := (D3 w.1, D3 w.2)
def enc4 (x : Reg) : C4 × C4 := (E4 x.1, E4 x.2)
def dec4 (w : C4 × C4) : Option Reg := do
  let a ← D4 w.1
  let b ← D4 w.2
  pure (a,b)

def valid4 (w : C4 × C4) : Prop :=
  (∃ a, H4 w.1 (E4 a) ≤ 1) ∧ (∃ b, H4 w.2 (E4 b) ≤ 1)

def T3 (w : C3 × C3) := enc3 (update (dec3 w))
def T4 (w : C4 × C4) : Option (C4 × C4) := (dec4 w).map (fun x => enc4 (update x))

theorem register_roundtrip3 (x : Reg) : dec3 (enc3 x) = x := by
  rcases x with ⟨a,b⟩
  simp [dec3, enc3, roundtrip3]
theorem register_roundtrip4 (x : Reg) : dec4 (enc4 x) = some x := by
  rcases x with ⟨a,b⟩
  simp [dec4, enc4, roundtrip4]

theorem four_decoder_total_on_error_domain (w : C4 × C4) (h : valid4 w) :
    ∃ x, dec4 w = some x := by
  rcases h with ⟨⟨a,ha⟩,⟨b,hb⟩⟩
  refine ⟨(a,b), ?_⟩
  simp [dec4, one_error_corrected4 a w.1 ha, one_error_corrected4 b w.2 hb]

theorem decoder_domain_exact4 (w : C4) :
    (D4 w).isSome = true ↔ ∃ b, H4 w (E4 b) ≤ 1 := by
  rcases w with ⟨a,b,c,d⟩
  cases a <;> cases b <;> cases c <;> cases d <;> decide

theorem register_decoder_domain_exact4 (w : C4 × C4) :
    (dec4 w).isSome = true ↔ valid4 w := by
  have h1 := decoder_domain_exact4 w.1
  have h2 := decoder_domain_exact4 w.2
  cases ha : D4 w.1 <;> cases hb : D4 w.2 <;>
    simp_all [dec4, valid4]

theorem encoded4_valid (x : Reg) : valid4 (enc4 x) := by
  rw [← register_decoder_domain_exact4, register_roundtrip4]
  rfl

theorem encoder3_injective : Function.Injective enc3 := by
  intro x y h
  have hh := congrArg dec3 h
  simpa [register_roundtrip3] using hh

theorem encoder4_injective : Function.Injective enc4 := by
```

```
  intro x y h
  have hh := congrArg dec4 h
  simpa [register_roundtrip4] using hh

theorem update4_output_valid (w w' : C4 × C4) (h : T4 w = some w') :
    valid4 w' := by
  unfold T4 at h
  cases hd : dec4 w with
  | none => simp [hd] at h
  | some x =>
    have he : enc4 (update x) = w' := by simpa [hd] using h
    rw [← he]
    exact encoded4_valid (update x)

theorem encoded_action3 (x : Reg) : T3 (enc3 x) = enc3 (update x) := by
  simp [T3, register_roundtrip3]
theorem quotient_action3 (w : C3 × C3) : dec3 (T3 w) = update (dec3 w) := by
  simp [T3, register_roundtrip3]
theorem encoded_action4 (x : Reg) : T4 (enc4 x) = some (enc4 (update x)) := by
  simp [T4, register_roundtrip4]
theorem quotient_action4 (w : C4 × C4) :
    (T4 w).bind dec4 = (dec4 w).map update := by
  simp only [T4]
  cases h : dec4 w with
  | none => rfl
  | some x => simp [register_roundtrip4]

theorem preparation_encoding3 (x y : Reg) :
    dec3 (enc3 x) = x ∧ (fun (_ : C3 × C3) => enc3 x) (enc3 y) = enc3 x := by
  exact ⟨register_roundtrip3 x, rfl⟩
theorem preparation_encoding4 (x y : Reg) :
    dec4 (enc4 x) = some x ∧ (fun (_ : C4 × C4) => enc4 x) (enc4 y) = enc4 x := by
  exact ⟨register_roundtrip4 x, rfl⟩

/-- Negative control: two errors may land in the strict-majority tie domain. -/
theorem four_site_tie_rejected : D4 (false,false,false,false) = none := by decide

end Component
end Article85
/- ===== Components.lean: end ===== -/

/- ===== FiniteChecks.lean: start ===== -/

/-! Article 85 exact finite certificates. Integer native coordinates encode
(x,y,z)=(X/2,sqrt(3)*Y/2,0); squaredScaled is FOUR TIMES the
physical squared distance. All claims below state this integer kernel.
No exact point-group maximality or real embedding theorem is claimed. -/
namespace Article85
namespace FiniteChecks

abbrev P2 := Int × Int

def native : List P2 :=
  [(-2,2),(-1,3),(1,3),(2,2),(4,0),(5,-1),
   (5,-3),(4,-4),(2,-4),(1,-3),(-1,-1),(-2,0),
   (-1,1),(1,-1),(1,1),(2,-2),(2,0),(4,-2)]

def perimeter : List Nat := [0,1,2,3,14,16,4,5,17,6,7,8,9,15,13,10,11,12]
def ringEdges : List (Nat × Nat) :=
```

```
  perimeter.zip (perimeter.drop 1 ++ perimeter.take 1)
def bonds : List (Nat × Nat) := ringEdges ++ [(12,14),(13,16),(15,17)]
def point (n : Nat) : P2 := native[n]!
def squaredScaled (p q : P2) : Int :=
  (p.1-q.1)*(p.1-q.1)+3*(p.2-q.2)*(p.2-q.2)
def halfTurn (p : P2) : P2 := (3-p.1,-1-p.2)
def halfTurnIndex : List Nat := [6,7,8,9,10,11,0,1,2,3,4,5,17,16,15,14,13,12]
def hasBond (i j : Nat) : Bool := bonds.contains (i,j) || bonds.contains (j,i)

theorem native_cardinality : native.length = 18 ∧ native.Nodup := by decide
theorem native_edge_count : bonds.length = 21 ∧ bonds.Nodup := by decide
theorem native_unit_edges : bonds.all (fun e => squaredScaled (point e.1) (point e.2) == 4)
    = true := by decide

theorem native_centroid_sums : (native.map Prod.fst).sum = 27 ∧
    (native.map Prod.snd).sum = -9 := by decide

theorem native_diameter_kernel :
    (native.flatMap (fun p => native.map (squaredScaled p))).all (fun x => x ≤ 172) = true
    ∧
    squaredScaled (point 1) (point 7) = 172 := by decide

theorem native_halfturn_points : (List.range 18).all (fun i =>
    halfTurn (point i) == point (halfTurnIndex[i]!)) = true := by decide

theorem native_halfturn_bonds : bonds.all (fun e =>
    hasBond (halfTurnIndex[e.1]!) (halfTurnIndex[e.2]!)) = true := by decide

def dx : P2 := ((point 8).1 - (point 2).1, (point 8).2 - (point 2).2)
def dy : P2 := ((point 5).1 - (point 11).1, (point 5).2 - (point 11).2)

theorem ordered_port_differences : dx = (1,-7) ∧ dy = (7,-1) := by decide
theorem ordered_port_metric_kernel :
    dx.1*dx.1+3*dx.2*dx.2 = 148 ∧
    dy.1*dy.1+3*dy.2*dy.2 = 52 ∧
    dx.1*dy.1+3*dx.2*dy.2 = 28 ∧
    dx.1*dy.2-dx.2*dy.1 = 48 := by decide

theorem ordered_ports_not_orthogonal : dx.1*dy.1+3*dx.2*dy.2 ≠ 0 := by decide

def primaryDualPairs : List (Nat × Nat) := [(2,1),(5,4),(8,7),(11,10)]
theorem primary_dual_backward_steps : primaryDualPairs.all (fun e =>
    ringEdges.contains (e.2,e.1)) = true := by decide

structure I3 where
  x : Int
  y : Int
  z : Int
  deriving DecidableEq, Repr

def add3 (a b : I3) : I3 := ⟨a.x+b.x,a.y+b.y,a.z+b.z⟩
def box : List I3 := [-1,0,1].flatMap fun x => [-1,0,1].flatMap fun y =>
  [-1,0,1].map fun z => ⟨x,y,z⟩
def directions : List I3 := [⟨1,0,0⟩,⟨-1,0,0⟩,⟨0,1,0⟩,
  ⟨0,-1,0⟩,⟨0,0,1⟩,⟨0,0,-1⟩]
def positiveDirections : List I3 := [⟨1,0,0⟩,⟨0,1,0⟩,⟨0,0,1⟩]
def boxBoundary := box.flatMap fun k => directions.filterMap fun d =>
  if box.contains (add3 k d) then none else some (k,d)
def positiveEdges := box.flatMap fun k => positiveDirections.filterMap fun d =>
```

```
  if box.contains (add3 k d) then some (k,d) else none

def boxInterior := box.filter fun k => directions.all fun d => box.contains (add3 k d)

theorem box_counts : box.length = 27 ∧ positiveEdges.length = 54 ∧
    boxBoundary.length = 54 ∧ box.Nodup := by decide

theorem box_interior : boxInterior = [⟨0,0,0⟩] := by decide

theorem box_boundary_identity : boxBoundary.length + 2 * positiveEdges.length = 6 * box.
    length := by decide

theorem box_native_inventory : box.length * native.length = 486 ∧
    box.length * bonds.length = 567 ∧ box.length * 26 = 702 ∧
    box.length * 8 = 216 := by decide

/-- Exact count formula for the distinct seven-cell operating specialization;
B is the processing-event count, K the number of assembled components. -/
def carbonIn (B K : Int) := 81314 + 44*K + 264*B
def carbonOut (B K : Int) := 22*(13000-10*B)+22*(3000-2*B)+44*(2-K)

theorem seven_cell_carbon_conservation (B K : Int) :
    carbonIn B K + carbonOut B K = 433402 := by
  unfold carbonIn carbonOut
  omega

theorem seven_cell_inventory_guards (B D K : Int)
    (hb : 0 ≤ B) (hmax : B ≤ 1000) (hd : 0 ≤ D) (hdb : D ≤ B)
    (hk1 : 1 ≤ K) (hk2 : K ≤ 2) :
    3000 ≤ 13000-10*B ∧ 1000 ≤ 3000-2*B ∧
    0 ≤ D ∧ 0 ≤ B-D ∧ 0 ≤ 44*(2-K) := by omega

abbrev Formula := Nat × Nat × Nat × Nat

def donor : Formula := (22,20,0,4)
def ammonia : Formula := (0,3,1,0)
def tetramer : Formula := (8,14,4,5)
def glycine : Formula := (2,5,1,2)
def remnant : Formula := (20,16,0,2)
def hydrogen : Formula := (0,2,0,0)
def water : Formula := (0,2,0,1)
def scale (n : Nat) (f : Formula) : Formula :=
  (n*f.1,n*f.2.1,n*f.2.2.1,n*f.2.2.2)
def sumFormula (f g : Formula) : Formula :=
  (f.1+g.1,f.2.1+g.2.1,f.2.2.1+g.2.2.1,f.2.2.2+g.2.2.2)
def inputFormula := sumFormula (scale 12 donor) (scale 10 ammonia)
def outputFormula := [scale 2 tetramer,scale 2 glycine,scale 10 remnant,
  scale 2 donor,scale 10 hydrogen,scale 6 water].foldl sumFormula (0,0,0,0)

theorem packet_element_balance : inputFormula = (264,270,10,48) ∧
    outputFormula = (264,270,10,48) := by decide

/-- Same immutable atom universe and element/origin functions do not force
same current bond graph. This is a counterexample, not an atom-map model. -/
def intact : Bool → Bool → Bool := fun a b => a != b
def severed : Bool → Bool → Bool := fun _ _ => false

theorem ancestry_does_not_preserve_bonds : intact ≠ severed := by
  intro h
```

```
  have he := congrFun (congrFun h false) true
  contradiction

/-- Rational support constants only: no differential-inequality or event-clock
existence theorem is encoded by these arithmetic identities. -/
def availability : Rat := (30/31)*(100/101)*(19/20)
def lowerGain (p : Rat) : Rat := availability * (1 + (99/50)*p/(1+p))-p

theorem availability_exact : availability = (2850/3131 : Rat) := by decide +kernel

theorem gain_endpoint_exact : lowerGain (81/50) = (8298609/20508050 : Rat) ∧
    0 < lowerGain (81/50) := by decide +kernel

theorem threshold_time_arithmetic :
    (161/100 : Rat) / lowerGain (81/50) = 66035921/16597218 ∧
    (66035921/16597218 : Rat) < 4 := by decide +kernel

theorem processing_floor_arithmetic :
    (1000 : Rat) ≤ 10*4*(5030/201) ∧ 10*4*(5030/201) < (1001 : Rat) := by decide +kernel

theorem water_upper_barrier_arithmetic :
    (54/590 : Rat)+4/(590*590)-1/10 = -1473/174050 ∧
    (54/590 : Rat)+4/(590*590)-1/10 < 0 := by decide +kernel

theorem energy_lower_bound_arithmetic :
    (160 : Rat)-100-4*(2+151/50)-18 = 548/25 ∧ (0 : Rat) < 548/25 := by decide +kernel

theorem capsule_margin_arithmetic :
    (36 : Rat)*(22/7)*201*201 < 500*500*500 ∧ (4/5 : Rat) < 402/501 := by decide +kernel

theorem common_extent_amounts (xi : Rat) (h0 : 0 ≤ xi) (h1 : xi < 1) :
    0 < 1-xi ∧ (1-xi)+xi = 1 ∧ 500 ≤ 500+xi ∧ 500+xi < 501 := by
  grind


end FiniteChecks
end Article85
/- ===== FiniteChecks.lean: end ===== -/

/- ===== Countermodels.lean: start ===== -/

/-! Explicit obstructions to tempting stronger statements. -/
namespace Article85
namespace Countermodels

inductive C3 where
  | a | b | c
  deriving DecidableEq

def cyclicStep : C3 → C3 → Int
  | .a, .b => 1
  | .b, .c => 1
  | .c, .a => 1
  | .b, .a => -1
  | .c, .b => -1
  | .a, .c => -1
  | _, _ => 0

def windingGraph : LabelledGraph C3 where
```

```
    edge := fun x y => x ≠ y
    symmetric := fun h he => h he.symm
    step := fun x y i => if i.val = 0 then cyclicStep x y else 0
    antisymmetric := by
      intro x y _ i
      cases x <;> cases y <;> by_cases hi : i.val = 0 <;> simp [cyclicStep, hi]

def windingWalk : Walk windingGraph .a .a :=
  .cons (y := .b) (by change C3.a ≠ C3.b; decide) (.cons (y := .c) (by change C3.b ≠ C3.c;
    decide) (.cons (y := .a) (by change C3.c ≠ C3.a; decide) (.nil _)))

theorem winding_integral_nonzero : windingWalk.integral 0 = 3 := by rfl

theorem winding_has_no_coordinate : ¬ ∃ a, Coordinate windingGraph a := by
  rintro ⟨a,ha⟩
  have hz := coordinate_zero_periods windingGraph a ha C3.a windingWalk 0
  rw [winding_integral_nonzero] at hz
  contradiction

def zero3 : Vec3 := fun _ => 0
def plusX : Vec3 := fun i => if i.val = 0 then 1 else 0
def minusX : Vec3 := fun i => if i.val = 0 then -1 else 0
def singletonRegion (k : Vec3) : Prop := k 0 = 0 ∧ k 1 = 0 ∧ k 2 = 0
instance : DecidablePred singletonRegion := fun k => inferInstanceAs (Decidable (k 0 = 0 ∧ k
     1 = 0 ∧ k 2 = 0))

theorem missing_intermediate_changes_domain :
    (partialTranslate singletonRegion plusX zero3).bind
      (partialTranslate singletonRegion minusX) = none ∧
    partialTranslate singletonRegion zero3 zero3 = some zero3 := by
  simp [partialTranslate, singletonRegion, zero3, plusX]
  rfl

/-- A missing nearest-neighbor incidence cannot be repaired by coordinates
alone. Two coordinate-distinct cells at unit distance may have no edge. -/
def missingEdge : LabelledGraph Bool where
  edge := fun _ _ => False
  symmetric := False.elim
  step := fun _ _ _ => 0
  antisymmetric := False.elim

def separatedPoints (b : Bool) : Vec3 := if b then plusX else zero3

theorem coordinate_without_completeness : Coordinate missingEdge separatedPoints ∧
    ¬ NeighborComplete missingEdge separatedPoints := by
  constructor
  · intro _ _ h
    exact h.elim
  · intro hc
    have hn : IsUnitStep (fun i => separatedPoints true i - separatedPoints false i) := by
      refine ⟨0, Or.inl (by rfl), ?_⟩
      intro j hj
      have hval : j.val ≠ 0 := by
        intro he
        apply hj
        exact Fin.ext he
      simp [separatedPoints, plusX, zero3, hval]
    exact hc false true hn
```

```
end Countermodels
end Article85
/- ===== Countermodels.lean: end ===== -/

/- ===== Material.lean: start ===== -/

/-! Formal current-material editing kernel: fixed atom universe, immutable
labels and selected current-bond protection. No valence/reaction grammar or
complete endpoint atom map is manufactured from elemental counts. -/
namespace Article85
namespace Material

structure State (Atom Element Origin Extra : Type) where
  element : Atom → Element
  origin : Atom → Origin
  currentBond : Atom → Atom → Nat
  extra : Extra

def edit {A E O X : Type} (selected : A → A → Bool)
    (target : A → A → Nat) (s : State A E O X) : State A E O X :=
  { s with currentBond := fun a b => if selected a b then target a b else s.currentBond a b }


theorem editing_preserves_element {A E O X : Type} (selected : A → A → Bool)
    (target : A → A → Nat) (s : State A E O X) :
    (edit selected target s).element = s.element := by rfl

theorem editing_preserves_origin {A E O X : Type} (selected : A → A → Bool)
    (target : A → A → Nat) (s : State A E O X) :
    (edit selected target s).origin = s.origin := by rfl

theorem editing_preserves_protected_bond {A E O X : Type}
    (selected : A → A → Bool) (keepBond : A → A → Prop)
    (hdisjoint : ∀ a b, keepBond a b → selected a b = false)
    (target : A → A → Nat) (s : State A E O X) (a b : A) (hp : keepBond a b) :
    (edit selected target s).currentBond a b = s.currentBond a b := by
  simp [edit, hdisjoint a b hp]

/-- Generic composition theorem instantiated to exact current keepBond bonds. -/
theorem legal_word_preserves_current_bonds {A E O X U : Type}
    (ops : PartialActions U (State A E O X)) (keepBond : A → A → Prop)
    (h : ∀ u s t, ops u s = some t → ∀ a b, keepBond a b →
      t.currentBond a b = s.currentBond a b)
    (word : List U) (s t : State A E O X)
    (legal : executeWord ops word s = some t) :
    ∀ a b, keepBond a b → t.currentBond a b = s.currentBond a b := by
  intro a b hp
  exact word_conserves_quantity ops (fun x => x.currentBond a b)
    (fun u x y hh => h u x y hh a b hp) word s t legal

end Material
end Article85
/- ===== Material.lean: end ===== -/

/- Exact per-theorem kernel-axiom reports. -/
#print axioms Article85.Walk.integral_append
#print axioms Article85.Walk.integral_reverse
#print axioms Article85.integral_telescope
#print axioms Article85.coordinate_zero_periods
```

```
#print axioms Article85.zero_periods_coordinate
#print axioms Article85.recognition_existence
#print axioms Article85.coordinate_unique_translation
#print axioms Article85.coordinate_injective_iff
#print axioms Article85.induced_incidence_iff
#print axioms Article85.recognition_full
#print axioms Article85.translation_compose_on_domain
#print axioms Article85.intertwining_all_words
#print axioms Article85.intertwining_identity
#print axioms Article85.intertwining_compose
#print axioms Article85.intertwining_legality_iff
#print axioms Article85.quotient_lifts_word
#print axioms Article85.encoded_word_output
#print axioms Article85.intertwining_observed_trace
#print axioms Article85.word_preserves_predicate
#print axioms Article85.word_conserves_quantity
#print axioms Article85.independent_guarded_updates_commute
#print axioms Article85.disjoint_writes_can_change_domain
#print axioms Article85.one_step_decoding_is_not_iteration
#print axioms Article85.shared_update_exists_iff
#print axioms Article85.shared_update_unique
#print axioms Article85.shared_common_update
#print axioms Article85.shared_lifts_compatible
#print axioms Article85.shared_partial_update
#print axioms Article85.two_stock_flux_cancels
#print axioms Article85.scalar_schur_elimination
#print axioms Article85.path_terminal_current
#print axioms Article85.cycle_terminal_current
#print axioms Article85.midpoint_kirchhoff
#print axioms Article85.midpoint_unique
#print axioms Article85.contextual_terminal_replacement
#print axioms Article85.complement_involution
#print axioms Article85.reversal_involution
#print axioms Article85.complement_reversal_commute
#print axioms Article85.toggleF_involution
#print axioms Article85.toggleR_involution
#print axioms Article85.reference_actions_commute
#print axioms Article85.normalWord_F
#print axioms Article85.normalWord_R
#print axioms Article85.normalWord_action
#print axioms Article85.action_preserves_resident
#print axioms Article85.action_involution
#print axioms Article85.reaches_normal_form
#print axioms Article85.zero_tag_representative_unique
#print axioms Article85.refAction_compose
#print axioms Article85.orbit_iff_normal
#print axioms Article85.pairBase_bits
#print axioms Article85.baseBits_base
#print axioms Article85.codon_roundtrip
#print axioms Article85.word_roundtrip
#print axioms Article85.pairBase_reversal
#print axioms Article85.transported_reversal
#print axioms Article85.selected_initializations
#print axioms Article85.selected_codons
#print axioms Article85.initState_domain
#print axioms Article85.dispatch_reference_invariant
#print axioms Article85.preparation_is_initialization
#print axioms Article85.register_update_nonidentity
#print axioms Article85.register_update_involution
```

```
#print axioms Article85.initialization_distinct_from_update
#print axioms Article85.operation_preserves_population
#print axioms Article85.operation_preserves_address
#print axioms Article85.allWords_complete
#print axioms Article85.address_enumeration_counts
#print axioms Article85.referenced_normal_form_count
#print axioms Article85.coarser_quotient_counts
#print axioms Article85.Component.E3_distance
#print axioms Article85.Component.E4_distance
#print axioms Article85.Component.one_error_corrected3
#print axioms Article85.Component.one_error_corrected4
#print axioms Article85.Component.roundtrip3
#print axioms Article85.Component.roundtrip4
#print axioms Article85.Component.balls_disjoint3
#print axioms Article85.Component.balls_disjoint4
#print axioms Article85.Component.register_roundtrip3
#print axioms Article85.Component.register_roundtrip4
#print axioms Article85.Component.four_decoder_total_on_error_domain
#print axioms Article85.Component.decoder_domain_exact4
#print axioms Article85.Component.register_decoder_domain_exact4
#print axioms Article85.Component.encoded4_valid
#print axioms Article85.Component.encoder3_injective
#print axioms Article85.Component.encoder4_injective
#print axioms Article85.Component.update4_output_valid
#print axioms Article85.Component.encoded_action3
#print axioms Article85.Component.quotient_action3
#print axioms Article85.Component.encoded_action4
#print axioms Article85.Component.quotient_action4
#print axioms Article85.Component.preparation_encoding3
#print axioms Article85.Component.preparation_encoding4
#print axioms Article85.Component.four_site_tie_rejected
#print axioms Article85.FiniteChecks.native_cardinality
#print axioms Article85.FiniteChecks.native_edge_count
#print axioms Article85.FiniteChecks.native_unit_edges
#print axioms Article85.FiniteChecks.native_centroid_sums
#print axioms Article85.FiniteChecks.native_diameter_kernel
#print axioms Article85.FiniteChecks.native_halfturn_points
#print axioms Article85.FiniteChecks.native_halfturn_bonds
#print axioms Article85.FiniteChecks.ordered_port_differences
#print axioms Article85.FiniteChecks.ordered_port_metric_kernel
#print axioms Article85.FiniteChecks.ordered_ports_not_orthogonal
#print axioms Article85.FiniteChecks.primary_dual_backward_steps
#print axioms Article85.FiniteChecks.box_counts
#print axioms Article85.FiniteChecks.box_interior
#print axioms Article85.FiniteChecks.box_boundary_identity
#print axioms Article85.FiniteChecks.box_native_inventory
#print axioms Article85.FiniteChecks.seven_cell_carbon_conservation
#print axioms Article85.FiniteChecks.seven_cell_inventory_guards
#print axioms Article85.FiniteChecks.packet_element_balance
#print axioms Article85.FiniteChecks.ancestry_does_not_preserve_bonds
#print axioms Article85.FiniteChecks.availability_exact
#print axioms Article85.FiniteChecks.gain_endpoint_exact
#print axioms Article85.FiniteChecks.threshold_time_arithmetic
#print axioms Article85.FiniteChecks.processing_floor_arithmetic
#print axioms Article85.FiniteChecks.water_upper_barrier_arithmetic
#print axioms Article85.FiniteChecks.energy_lower_bound_arithmetic
#print axioms Article85.FiniteChecks.capsule_margin_arithmetic
#print axioms Article85.FiniteChecks.common_extent_amounts
#print axioms Article85.Countermodels.winding_integral_nonzero
```

```
#print axioms Article85.Countermodels.winding_has_no_coordinate
#print axioms Article85.Countermodels.missing_intermediate_changes_domain
#print axioms Article85.Countermodels.coordinate_without_completeness
#print axioms Article85.Material.editing_preserves_element
#print axioms Article85.Material.editing_preserves_origin
#print axioms Article85.Material.editing_preserves_protected_bond
#print axioms Article85.Material.legal_word_preserves_current_bonds
```

8 All-declaration axiom audit

AuditAll.lean

```
import Lean
import Article85
/- This audit command creates no mathematical theorem. It audits every
namespace declaration, including generated eliminators and helper definitions. -/
open Lean Elab Command in
run_cmd do
  let env ← getEnv
  let entries := env.constants.toList.filter (fun (name, _) => Name.isPrefixOf `Article85
    name)
  let entries := entries.mergeSort (fun a b => Name.lt a.1 b.1)
  for (name, ci) in entries do
    let axioms ← collectAxioms name
    logInfo m!"AUDIT {name}|{axioms.qsort Name.lt |>.toList}"
    match ci with
    | .thmInfo _ => logInfo m!"THEOREM {name}"
    | _ => pure ()
  logInfo m!"AUDIT_COUNT {entries.length}"
```

9 Fresh compilation and rejection controls

audit_lean.py

```
#!/usr/bin/env python3
"""Fresh isolated Lean compilation, all-declaration audit, and live negative controls.
Tooling only: mathematical evidence is Lean's kernel-checked source, not Python.
No assert statements; runs identically under python3 and python3 -O.
"""
from pathlib import Path
import argparse,hashlib,json,os,re,shutil,subprocess,tempfile,time

ALLOWED={'propext','Classical.choice','Quot.sound'}
ROOT=Path(__file__).resolve().parent

def write_utf8(path,text):
    Path(path).write_text(text,encoding='utf-8')

def require(ok,msg):
    if not ok: raise RuntimeError(msg)

def clean_source(s):
    # Strip nested Lean block comments, line comments, and quoted strings.
    out=[]; i=0; depth=0
    while i<len(s):
        if depth:
```

```
            if s.startswith('/-',i): depth+=1; i+=2
            elif s.startswith('-/',i): depth-=1; i+=2
            else: i+=1
        elif s.startswith('/-',i): depth=1; i+=2
        elif s.startswith('--',i):
            j=s.find('\n',i); i=len(s) if j<0 else j
        elif s[i]=='"':
            i+=1
            while i<len(s):
                if s[i]=='\\': i+=2
                elif s[i]=='"': i+=1; break
                else: i+=1
            out.append(' ')
        else: out.append(s[i]); i+=1
    require(depth==0,'Unclosed source comment')
    return ''.join(out)

def static_check(source):
    clean=clean_source(source)
    forbidden=r'\b(sorry|admit|axiom|unsafe|native_decide|implemented_by|extern|ofReduceBool|ofReduceNat
    )\b'
    bad=re.findall(forbidden,clean)
    require(not bad,f'Forbidden proof/trust tokens: {bad}')
    imports=re.findall(r'^import (.+)$',clean,re.M)
    require(imports==['Std','Init.Data.Rat.Lemmas'],f'Unexpected imports: {imports}')
    require('\t' not in source,'Tab in self-contained source')
    require(source.endswith('\n') and not source.endswith('\n\n'),'Need exactly one terminal newline')
    require(all(line==line.rstrip() for line in source.splitlines()),'Trailing whitespace')

def run(cmd,cwd,log,env=None):
    start=time.monotonic()
    p=subprocess.run(cmd,cwd=cwd,env=env,stdout=subprocess.PIPE,stderr=subprocess.STDOUT,text=True,
     encoding='utf-8',timeout=240)
    write_utf8(Path(log),p.stdout)
    return {'exit':p.returncode,'seconds':round(time.monotonic()-start,3),'output':p.stdout}

def parse_audit(s):
    rows={}
    for name,values in re.findall(r'^AUDIT ([^|\n]+)\|\[([^\n]*)\]$',s,re.M):
        rows[name]=[v.strip() for v in values.split(',') if v.strip()]
    counts=re.findall(r'^AUDIT_COUNT (\d+)$',s,re.M)
    require(len(counts)==1 and len(rows)==int(counts[0]),'Incomplete all-declaration audit')
    disallowed={name:ax for name,ax in rows.items() if not set(ax)<=ALLOWED}
    return rows,disallowed

def main():
    ap=argparse.ArgumentParser()
    ap.add_argument('--lean',required=True)
    ap.add_argument('--output',default='evidence')
    ap.add_argument('--skip-mutations',action='store_true',help='Compile/audit only; status explicitly
     records skipped controls')
    args=ap.parse_args()
    found=shutil.which(args.lean) if '/' not in args.lean else args.lean
    require(found is not None, 'Lean executable not found')
    lean=str(Path(found).resolve()); out=Path(args.output).resolve()
    require(not out.exists() or not any(out.iterdir()), 'Output directory must be absent or empty;
     choose a fresh path')
    out.mkdir(parents=True,exist_ok=True)
    source=(ROOT/'Article85.lean').read_text(encoding='utf-8')
    static_check(source)
    version=subprocess.check_output([lean,'--version'],text=True,encoding='utf-8').strip()
    require('version 4.24.0,' in version and '797c613eb9b6d4ec95db23e3e00af9ac6657f24b' in version,'
     Wrong toolchain')
    write_utf8((out/'toolchain.txt'),version+'\n')
    manifest=json.loads((ROOT/'THEOREMS.json').read_text(encoding='utf-8'))
    require(manifest['count']==len(manifest['theorems']),'Theorem manifest inconsistent')
    printed=re.findall(r'^#print axioms (.+)$',source,re.M)
    require(printed==manifest['theorems'],'Per-theorem #print list differs from theorem manifest')
```

```
    with tempfile.TemporaryDirectory(prefix='article85-audit-') as td:
        work=Path(td)
        write_utf8((work/'Article85.lean'),source)
        (work/'AuditAll.lean').write_text((ROOT/'AuditAll.lean').read_text(encoding='utf-8'))
        env=dict(os.environ,LEAN_PATH=str(work))
        c=run([lean,'-j1','-M4500','-o','Article85.olean','Article85.lean'],work,out/'compile.log',env)
        require(c['exit']==0 and 'warning:' not in c['output'],'Fresh compilation failed or warned')
        for name in printed:
            require(f"'{name}' depends on axioms:" in c['output'] or f"'{name}' does not depend on any axioms" in c['output'],f'Missing #print axioms output: {name}')
        a=run([lean,'-j1','-M4500','AuditAll.lean'],work,out/'all_declarations.log',env)
        require(a['exit']==0,'All-declaration audit command failed')
        rows,disallowed=parse_audit(a['output'])
        require(not disallowed,f'Disallowed axioms: {disallowed}')
        require(set(printed)<=set(rows),'Authored theorem missing from environment audit')
        write_utf8((out/'axioms_by_declaration.json'),json.dumps(rows,indent=2,sort_keys=True)+'\n')
        controls=[]
        mutants=[
            ('carbon_total','carbonIn B K + carbonOut B K = 433402','carbonIn B K + carbonOut B K = 433403'),
            ('genetic_selected_word','def GCG : Word6 := \u27e8true,false,false,true,true,false\u27e9','def GCG : Word6 := \u27e8true,false,false,true,true,true\u27e9'),
            ('native_dot_product','dx.1*dy.1+3*dx.2*dy.2 = 28','dx.1*dy.1+3*dx.2*dy.2 = 0'),
            ('decoder_tie','if correctedOnes4 w = 2 then none else','if correctedOnes4 w = 2 then some false else'),
            ('preparation_action','| .prep x, _ => x','| .prep _, y => y'),
            ('winding_period','winding_integral_nonzero : windingWalk.integral 0 = 3',
              'winding_integral_nonzero : windingWalk.integral 0 = 0')]
        if not args.skip_mutations:
            def mutant_run(item):
                name,old,new=item
                require(source.count(old)==1,f'Mutant {name} anchor not unique')
                mdir=work/name; mdir.mkdir()
                write_utf8((mdir/'Mutant.lean'),source.replace(old,new,1))
                r=run([lean,'-j1','-M4500','Mutant.lean'],mdir,out/f'mutant_{name}.log',env)
                require(r['exit']!=0 and 'error:' in r['output'],f'Mutant {name} was not rejected with a Lean error; compiler exit={r["exit"]}')
                return {'name':name,'kind':'semantic mutation','rejected':True,'compile_exit':r['exit'],'seconds':r['seconds']}
            # Sequential compiler jobs avoid competing proof-elaboration memory peaks.
            controls=[mutant_run(item) for item in mutants]
            # Verify the trust audit rejects a compile-valid custom axiom.
            injected=source+'\naxiom Article85.auditInjection : False\n'
            try:
                static_check(injected)
                raise RuntimeError('Static custom-axiom detector failed')
            except RuntimeError as exc:
                require('Forbidden proof/trust tokens' in str(exc),'Unexpected static injection result')
            write_utf8((work/'Injected.lean'),injected)
            inj=run([lean,'-j1','-M4500','-o','Injected.olean','Injected.lean'],work,out/'injected_compile.log',env)
            require(inj['exit']==0,'Custom-axiom injection should compile before audit rejects it')
            aud=(ROOT/'AuditAll.lean').read_text(encoding='utf-8').replace('import Article85','import Injected')
            write_utf8((work/'AuditInjected.lean'),aud)
            ai=run([lean,'-j1','AuditInjected.lean'],work,out/'injected_axiom_audit.log',env)
            require(ai['exit']==0,'Injected audit failed unexpectedly')
            _,bad=parse_audit(ai['output'])
            require('Article85.auditInjection' in bad,'Live all-declaration detector missed custom axiom')
            controls.append({'name':'custom_axiom_injection','kind':'trust-boundary mutation','rejected':True,'compile_exit':0})
        result={'status':'PASS' if not args.skip_mutations else 'COMPILE_AUDIT_PASS_CONTROLS_SKIPPED',
          'source_sha256':hashlib.sha256(source.encode()).hexdigest(),'toolchain':version,
          'compile_exit':c['exit'],'all_axiom_audits_pass':True,'namespace_declaration_count':len(rows),
          'authored_named_theorem_count':len(printed),'allowed_foundations':sorted(ALLOWED),
          'used_foundations':sorted({ax for axs in rows.values() for ax in axs}),
          'negative_controls_pass':not args.skip_mutations and all(x['rejected'] for x in controls),
```

```
          'negative_controls':controls,'python_optimized':not __debug__}
        write_utf8((out/'audit_result.json'),json.dumps(result,indent=2)+'\n')
        print(json.dumps(result,indent=2))

if __name__=='__main__': main()
```

10 Complete recovery helper

```
#!/usr/bin/env python3
"""Recover the complete pinned Article 85 runtime; standard library only."""
from pathlib import Path, PurePosixPath
import argparse, base64, hashlib, io, json, stat, zipfile
EXPECTED_SHA256 = "de6888fa26fb84af3c6698d30f81304cb477cf3cfdbc289c43409ec2c2bbb8a9"
DEFAULT_PAYLOAD = "article85_formal_payload.b64"

def require(ok, message):
    if not ok:
        raise ValueError(message)

def recover(payload, output):
    encoded = payload.read_bytes()
    raw = base64.b64decode(b"".join(encoded.split()), validate=True)
    require(hashlib.sha256(raw).hexdigest() == EXPECTED_SHA256,
            "Payload archive digest mismatch")
    with zipfile.ZipFile(io.BytesIO(raw)) as z:
        infos = z.infolist()
        names = [i.filename for i in infos]
        require(len(names) == len(set(names)), "Duplicate archive member")
        for i in infos:
            p = PurePosixPath(i.filename)
            require(not p.is_absolute() and ".." not in p.parts
                    and str(p) == i.filename and "\\" not in i.filename
                    and ":" not in i.filename
                    and not i.is_dir(), "Unsafe archive member")
            require(not stat.S_ISLNK(i.external_attr >> 16), "Symbolic link")
            require(not (i.flag_bits & 1), "Encrypted archive member")
        manifest = json.loads(z.read("PAYLOAD_MANIFEST.json"))
        require(manifest["format"] == "article85-portable-payload-v1",
                "Unexpected manifest format")
        files = manifest["files"]
        require(set(names) == set(files) | {"PAYLOAD_MANIFEST.json"},
                "Archive inventory mismatch")
        contents = {}
        for n, row in files.items():
            data = z.read(n)
            require(len(data) == row["size"], "Size mismatch: " + n)
            require(hashlib.sha256(data).hexdigest() == row["sha256"],
                    "Member digest mismatch: " + n)
            contents[n] = data
        contents["PAYLOAD_MANIFEST.json"] = z.read("PAYLOAD_MANIFEST.json")
    require(not output.exists() or (output.is_dir() and
            not any(output.iterdir())), "Output directory must be empty")
    output.mkdir(parents=True, exist_ok=True)
    for n, data in contents.items():
        dest = output/n
        dest.parent.mkdir(parents=True, exist_ok=True)
        dest.write_bytes(data)
    print("Recovered", len(contents), "verified files into", output)

if __name__ == "__main__":
    ap = argparse.ArgumentParser(description=__doc__)
    ap.add_argument("--payload", type=Path,
                    default=Path(__file__).resolve().parent/DEFAULT_PAYLOAD)
    ap.add_argument("--output-dir", type=Path, required=True)
    args = ap.parse_args()
    recover(args.payload, args.output_dir)
```