Formal Proof Dossier

Article 75: Constructive Genetic Code-Table Realization On A Typed Chrysene Carrier

Charles D. Schaper, Ph.D.

Annals of the Chrysene Formalism, Vol. 1, No. 4
Manuscript pages 1503–1544

Verification supplement, 23 October 2026

Kernel-checked partial formalization

Proof source	`Article75.lean`
Tested toolchain	Lean 4.24.0
Imports	`Std`, `Init.Data.Rat.Lemmas`
Named theorems	**89**
Per-theorem dependency audits	**89 passed**
Custom axioms / proof holes	0 / 0
`native_decide` / unsafe declarations	0 / 0
Compiler exit status	0

Scope and interpretation

This is a **partial formalization**, not a proof of the entire manuscript. It proves selected finite registry, genetic-code incidence, routing, exact rational selection, graph-identity and controller/resource statements in the explicitly defined models below. The genetic-code tables are supplied reference data. No chemical synthesis, continuous molecular path, stereochemical selection, physical reaction rate, or unique origin of a genetic code is certified.

Reproduction and independent checks

The successful compiler was Lean 4.24.0, commit `797c613eb9b6d4ec95db23e3e00af9ac6657f24b`. No Mathlib dependency is needed. The complete UTF-8 Lean source is embedded below, *byte-for-byte identical* to the separately compiled file. To repeat the formal check:

```
lean Article75.lean
python3 audit.py /path/to/lean
```

The second command rebuilds the proof file, checks all 89 dependency reports, rejects proof holes and custom axioms, and checks the embedded-source identity when this document is present. It writes `build.log` and `audit_summary.json`. The source uses proof-producing tactics and kernel reduction, including `decide` and `decide +kernel`. It does not use `native_decide` or trust the Python verifier as a Lean theorem.

Overleaf: upload this single `.tex` file and compile with pdfLaTeX. No companion source, bibliography file, image, shell escape, or network access is needed to typeset it. Overleaf compiles the

document; it does *not* run Lean. Reproduce the proofs with the raw UTF-8 `Article75.lean`, rather than code copied from the PDF.

Compiled source SHA-256:
`ec9b4202578b3bc3a75e278dde3b4f82b20201500b424cb8e7c886402af1f083`

Concrete mathematical representation

A six-bit description is an element of `Fin 64`; its integer value uses the usual leading-bit order. Consecutive two-bit values $00, 01, 10, 11$ encode A, C, G, U, respectively. Thus a codon has index $16b_1 + 4b_2 + b_3$ with $A = 0, C = 1, G = 2, U = 3$. For example, AUG is 14, GCU is 39, UAA is 48, and UAC is 49. These are coordinates, not atom identities.

The global flip sends m to $63 - m$; reversal reverses all six bits. The orbit representative is the minimum of the four actual images. Adjacent differences are independently computed from the six bits. Equality of their reversal classes is proved equivalent to orbit equality for all 64^2 ordered pairs. The concrete quotient is the subtype of fixed representatives, so its factorization theorem applies to any target type and any function invariant under the two generators.

The three supplied 64-entry tables are the selected, vertebrate-mitochondrial and standard tables of the manuscript certificate. They are written explicitly in the source. The proof does not retrieve a live biological database or establish the transcription independently of those supplied data. The manuscript's separate verification supplement performs the broader archived-data and graph-path checks.

Coverage mapped to manuscript statements

The following are source labels in the manuscript, not external LaTeX references. Every listed Lean theorem is part of the complete source. A checked computational certificate is distinguished from a general theorem not formalized here.

Group and manuscript anchor	Kernel-checked result	Explicit boundary
F1: `reg: transport`, `reg: complete`, `reg: count`	Inverse encoding; commuting involutions; transported reversal; exact difference fibres and complete invariant; canonical representatives; 20 orbits, eight of size 2 and twelve of size 4; eight reversal and eight flip-reversal fixed points; palindromic-difference criterion; unique factorization of any invariant output.	This is the six-bit $N = 3$ instance. The all-N orbit formula, general cardinality capacity chain, saturation theorem and selector/partition classification are not formalized.

Anchor	Kernel-checked result	Explicit boundary
F2: `reg:tether`	Normalization is invariant under both decorated actions; zero-tag representatives are unique; residual marker quotients have 32, 36 and 36 representatives. The normalized slice ranges over all 64 markers.	The resident AAA tether is not an external variable; no chemical reference-tag action or physical preparation is proved. No general quotient library is used.
F3: `reg:matching`, `reg:parity`, `reg:refinement`	Valid distinct matching witnesses of sizes 18 (WC), 19 (RY), 20 (MK); the displayed 9-to-7 and 9-to-8 Hall neighborhoods; singleton Met/Trp and three-member STOP fibres; a direct impossibility of factoring the standard table through the registry; exactly 61 registry/output keys and exactly the three reported doublets; an injection of `Fin 61` into every state space retaining both values.	The sharp matching upper bounds require the general counting argument in the manuscript: this file checks the obstruction data, not that general argument. The all-24-encoding classification and the general even-fibre theorem are not formalized. The 61-state lower bound, in contrast, is proved as a genuine injection theorem for arbitrary target types.
F4: `reach:counts`, `reach:two-swaps`, `reach:full-path`, `reach:robust`	The 24 cells form eight quartets and sixteen doublets; the selected table has 20 sense labels; both chosen exchanges satisfy the declared finite locality/lock/occupation guards; their composition equals the mitochondrial table at all 64 codons; four endpoint differences, common AUG/GCU/UAA tape, and 63 distinct endpoint sense pairs.	This does not prove the full seed/propagation/refinement/phase execution, the 840 SDR count, the minimum number of exchanges, or the large routing-orbit theorem. The guard is checked for the two specified moves, not a complete implementation of all states.
F5: `reach:nested-no-go`, `reach:crossing`, `reach:full-path`	Actual positive-maximizer selection from finite branch lists; universal rational regimes below and above 1/2, the exact tie at 1/2, both all-codon endpoint readings, persistent AUG and UAA; nested-singleton obstruction; decreasing-score positivity and order crossing at 2/3.	Control is over exact rationals, not a formalized real-continuous path. The maximum-bottleneck continuity theorem, compactness and geometric lifts are excluded. The coefficients and branch destinations remain supplied model data.
F6: `junc:word`, `junc:truth`	Directed incoming/outgoing complement relation, all four release rows, the AUG boundary witness, impossibility of the untransported paired-width gate, uniqueness of exact register transport.	No physical release mechanism, selection of this family, general frame-covariance theorem, sulfur delivery or stereochemistry follows.

Anchor	Kernel-checked result	Explicit boundary
F7: `junc:identity`	Explicit bijection between the nine source heavy atoms and the conventional Met template; preservation of every element, every ordered-pair bond order and the amino-acid root. The small decoder recognizes this graph as Met.	This is one rooted heavy-atom constitutional graph. It does not replay its synthesis, map hydrogens, check Cys or the other 18 products, or establish stereochemistry.
F8: `junc:metfirst`	A word/phase/position controller projection is defined by exact constructors. Every finite run from its idle state has Met-first nonempty archived words; active/stopped/released words have the same first identity, a set phase and position divisible by three. Generic path composition and invariant/projection propagation are proved by induction.	START uses the computed graph identity from F7, not a Met equality hypothesis. The projection omits atom ownership, packet IDs, token ledger and certificate guards; no formal simulation from the full implementation is proved. This is not the complete four-part manuscript controller theorem.
F9: `chem:edit-word`, `junc:copying`, `lem:disjoint-composition`, `lem:interface-composition`	Injective origin-prefix copying and disjoint copies; split/join endpoint-capacity conservation with explicit guards; bond-order limit; commutation of distinct fixed-coordinate updates; permanently consumed tokens under every finite resource run; additive costs, prefix budgets and $R_0 + nR_B$.	These are local bookkeeping and generic resource lemmas, not the full guarded bond-word theorem or a compositional proof for the complete molecular/controller state. A resource-run theorem is not a full proof that the controller implementation refines that run.

Important qualifications

1. **No imported conclusion is an axiom.** The finite action, tables, branch scores and graph templates are definitions. Computational claims reduce them explicitly. Generic preservation and factorization results state their hypotheses. The audit permits only Lean's standard foundations `propext`, `Classical.choice` and `Quot.sound`; "no custom axioms" is not a claim of no foundational axioms.
2. **A data-conditioned table theorem is not a derivation of the genetic code.** The reference tables and routing/selection instance are supplied inputs. The verified equalities and obstructions concern those exact inputs. Product labels and nucleobase coordinates have different types.
3. **The 61-state bound is not merely a count.** The list of 61 representative descriptions has pairwise different registry/output keys. If maps u, v recover both keys from a state map z, then $i \mapsto z(m_i)$ is injective. Thus every such state space contains at least 61 distinguishable states. The directly checked pair image attains 61. This conclusion does not assert physical realization of those states.
4. **Projected controller safety is not full-machine certification.** The inductive proof covers every run of the displayed projection, not only a bounded tape. It does not prove that every

execution of the separate Python controller or full manuscript tuple projects to it. Missing full-guard, origin-ledger and token-correspondence obligations remain explicit. In particular, an identity list is not a covalent peptide graph.

5. **The product-grammar certificate is separate.** This Lean file does not recheck the 20 diagnostic graph words, 174 paired sulfur edits, extraction-mask audit, global atom/port inventory, hydrogen assignment, or all target identities. The nine-heavy-atom Met comparison is a genuine finite graph-isomorphism check, but is not a substitute for those wider computations.
6. **No complete joint-realization or molecular-transfer theorem is claimed.** The real-continuous, geometric, chemical and stereochemical lifting assumptions are not established. This dossier must not be described as a formal proof of the whole manuscript or a machine proof of chemical synthesis.

Selected exact results

The concrete six-bit quotient has $20 = 8 + 12$ classes, with total descriptions $8 \cdot 2 + 12 \cdot 4 = 64$. The registry/output refinement has $61 = 64 - 3$ cells; its only doublets are AGU/UCA, AGC/UCG and CUU/UUG. The two chosen address exchanges reach the mitochondrial table without altering AUG, GCU or UAA.

For the supplied selector, the AUA score difference is

$$(1 + 3u) - (2 + u) = 2u - 1.$$

Below $u = 1/2$, AUA selects Met, UGA selects Trp and AGR is absent. At the wall, AUA selects both Ile and Met, while UGA and AGR are absent. Above it, AUA selects Ile, UGA is absent and AGR selects Arg. These are proved for all rational control values in the stated regimes, not sampled values. Absence is represented by an empty list and is not itself a controller halt.

Complete Lean 4 source

All definitions, all 89 named theorems, and all 89 dependency-audit commands follow. Typographic Unicode substitutions in the listing do not alter the underlying UTF-8 source embedded in this standalone file.

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

/-!
Article 75: partial, kernel-checked finite and rational formalization.
Reference title: Constructive Genetic Code-Table Realization On A Typed
Chrysene Carrier. The tables below are explicit supplied data, not predictions.
No chemical realization, stereochemistry, kinetics, or full graph-path replay
is asserted. See the accompanying dossier for theorem-level coverage.
-/
namespace Article75
set_option maxRecDepth 1000000
set_option maxHeartbeats 0

-- F1. Actual six-bit action; leading bits encode A=00,C=01,G=10,U=11.
abbrev Marker := Fin 64
instance : Min Marker := ⟨fun a b => if a.val ≤ b.val then a else b⟩
abbrev Base := Fin 4
abbrev Word := Base × Base × Base

```

```
def mkMarker (n : Nat) : Marker := ⟨n % 64, Nat.mod_lt _ (by decide)⟩
def mkBase (n : Nat) : Base := ⟨n % 4, Nat.mod_lt _ (by decide)⟩
def bit (m : Marker) (i : Nat) : Nat := m.val / 2^i % 2
def flip (m : Marker) : Marker := ⟨63-m.val, by omega⟩
def reverseMarker (m : Marker) : Marker := mkMarker
  (32*bit m 0 + 16*bit m 1 + 8*bit m 2 + 4*bit m 3 + 2*bit m 4 + bit m 5)
def encode (m : Marker) : Word :=
  (mkBase (m.val/16), mkBase (m.val/4), mkBase m.val)
def decode (w : Word) : Marker := mkMarker (16*w.1.val + 4*w.2.1.val + w.2.2.val)
def complement (b : Base) : Base := ⟨3-b.val, by omega⟩
def pairSwap (b : Base) : Base := mkBase (2*(b.val%2) + b.val/2)
def wordFlip (w : Word) : Word :=
  (complement w.1, complement w.2.1, complement w.2.2)
def wordReverse (w : Word) : Word := (w.2.2, w.2.1, w.1)
def transportedReverse (w : Word) : Word :=
  (pairSwap w.2.2, pairSwap w.2.1, pairSwap w.1)

theorem encoding_left_inverse : ∀ m : Marker, decode (encode m) = m := by decide
theorem encoding_right_inverse : ∀ a b c : Base, encode (decode (a,b,c)) = (a,b,c) := by decide
theorem flip_involution : ∀ m : Marker, flip (flip m) = m := by decide
theorem reverse_involution : ∀ m : Marker, reverseMarker (reverseMarker m) = m := by decide
theorem generators_commute : ∀ m : Marker,
    flip (reverseMarker m) = reverseMarker (flip m) := by decide
theorem flip_has_no_fixed_point : ∀ m : Marker, flip m ≠ m := by decide
theorem encoding_flip_transport : ∀ m : Marker, encode (flip m) = wordFlip (encode m) := by decide
theorem encoding_reverse_transport : ∀ m : Marker,
    encode (reverseMarker m) = transportedReverse (encode m) := by decide
theorem reversal_is_not_ordinary :
    transportedReverse (encode 49) ≠ wordReverse (encode 49) := by decide

def sameOrbit (m n : Marker) : Prop :=
  n = m ∨ n = flip m ∨ n = reverseMarker m ∨ n = flip (reverseMarker m)
instance (m n : Marker) : Decidable (sameOrbit m n) := inferInstanceAs (Decidable (_ ∨ _ ∨ _ ∨ _))
def orbit (m : Marker) : List Marker :=
  [m, flip m, reverseMarker m, flip (reverseMarker m)] |>.eraseDups
def representative (m : Marker) : Marker :=
  min (min m (flip m)) (min (reverseMarker m) (flip (reverseMarker m)))
def difference (m : Marker) : Nat :=
  16*((bit m 5 + bit m 4)%2) + 8*((bit m 4 + bit m 3)%2) +
  4*((bit m 3 + bit m 2)%2) + 2*((bit m 2 + bit m 1)%2) +
  (bit m 1 + bit m 0)%2
def reverseDifference (d : Nat) : Nat :=
  16*(d%2) + 8*(d/2%2) + 4*(d/4%2) + 2*(d/8%2) + d/16%2
def differenceClass (m : Marker) : Nat := min (difference m) (reverseDifference (difference m))
def allMarkers := List.finRange 64
def representatives := (allMarkers.map representative).eraseDups

theorem enumeration_complete : ∀ m : Marker, m ∈ allMarkers := by decide
theorem difference_fibres : ∀ m n : Marker,
    difference m = difference n ↔ n = m ∨ n = flip m := by decide
theorem difference_transport : ∀ m : Marker,
    difference (flip m) = difference m ∧
    difference (reverseMarker m) = reverseDifference (difference m) := by decide
theorem complete_invariant_six_bits : ∀ m n : Marker,
    sameOrbit m n ↔ differenceClass m = differenceClass n := by decide
theorem representative_classification : ∀ m n : Marker,
    representative m = representative n ↔ sameOrbit m n := by decide
theorem representative_in_orbit : ∀ m : Marker, sameOrbit m (representative m) := by decide
theorem representative_idempotent : ∀ m : Marker,
    representative (representative m) = representative m := by decide
theorem twenty_orbits : representatives.length = 20 := by decide
theorem orbit_size_dichotomy : ∀ m : Marker, (orbit m).length = 2 ∨ (orbit m).length = 4 := by decide
theorem orbit_size_distribution :
    (representatives.filter (fun m => (orbit m).length == 2)).length = 8 ∧
    (representatives.filter (fun m => (orbit m).length == 4)).length = 12 := by decide
theorem fixed_point_counts :
    (allMarkers.filter (fun m => reverseMarker m == m)).length = 8 ∧
    (allMarkers.filter (fun m => flip (reverseMarker m) == m)).length = 8 := by decide
theorem short_orbit_iff_palindrome : ∀ m : Marker,
```

```
    (orbit m).length = 2 ↔ difference m = reverseDifference (difference m) := by decide

-- Factorization on the concrete quotient, not an assumed quotient conclusion.
abbrev Registry := {m : Marker // representative m = m}
def quotientMap (m : Marker) : Registry := ⟨representative m, representative_idempotent m⟩

theorem invariant_on_orbit {Y : Type} (h : Marker → Y)
    (hf : ∀ m, h (flip m) = h m) (hr : ∀ m, h (reverseMarker m) = h m)
    {m n : Marker} (rel : sameOrbit m n) : h n = h m := by
  rcases rel with hmn | hmn | hmn | hmn
  · rw [hmn]
  · rw [hmn, hf]
  · rw [hmn, hr]
  · rw [hmn, hf, hr]

theorem invariant_factors_uniquely {Y : Type} (h : Marker → Y)
    (hf : ∀ m, h (flip m) = h m) (hr : ∀ m, h (reverseMarker m) = h m) :
    ∃ k : Registry → Y, (∀ m, k (quotientMap m) = h m) ∧
      ∀ k' : Registry → Y, (∀ m, k' (quotientMap m) = h m) → k' = k := by
  refine ⟨fun r => h r.val, ?_, ?_⟩
  · intro m
    exact invariant_on_orbit h hf hr (representative_in_orbit m)
  · intro k' hk
    funext r
    have eqr : quotientMap r.val = r := Subtype.ext r.property
    simpa only [eqr] using hk r.val

-- F2. Reference tags act on descriptions, never on resident attachments.
structure Addressed where
  marker : Marker
  reference : Bool
  direction : Bool
  deriving DecidableEq

def actF (s : Addressed) : Addressed := ⟨flip s.marker, !s.reference, s.direction⟩
def actR (s : Addressed) : Addressed := ⟨reverseMarker s.marker, s.reference, !s.direction⟩
def normalize (s : Addressed) : Marker :=
  let m := if s.direction then reverseMarker s.marker else s.marker
  if s.reference then flip m else m

theorem normalized_action : ∀ m : Marker, ∀ a d : Bool,
    normalize (actF ⟨m,a,d⟩) = normalize ⟨m,a,d⟩ ∧
    normalize (actR ⟨m,a,d⟩) = normalize ⟨m,a,d⟩ := by decide
theorem normalized_zero_tags : ∀ m : Marker, normalize ⟨m,false,false⟩ = m := by decide
theorem normalized_representatives_unique : ∀ m n : Marker,
    normalize ⟨m,false,false⟩ = normalize ⟨n,false,false⟩ ↔ m = n := by decide
theorem residual_counts :
    ((allMarkers.map (fun m => min m (flip m))).eraseDups).length = 32 ∧
    ((allMarkers.map (fun m => min m (reverseMarker m))).eraseDups).length = 36 ∧
    ((allMarkers.map (fun m => min m (flip (reverseMarker m)))).eraseDups).length = 36 := by decide

-- F3. Supplied code tables. Names are amino-acid identities, not nucleobases.
inductive Residue where
  | A | C | D | E | F | G | H | I | K | L | M | N | P | Q | R | S | T | V | W | Y | stop
  deriving DecidableEq, Repr, Inhabited
open Residue

def standardData : List Residue := [
  .K, .N, .K, .N, .T, .T, .T, .T, .R, .S, .R, .S, .I, .I, .M, .I,
  .Q, .H, .Q, .H, .P, .P, .P, .P, .R, .R, .R, .R, .L, .L, .L, .L,
  .E, .D, .E, .D, .A, .A, .A, .A, .G, .G, .G, .G, .V, .V, .V, .V,
  .stop, .Y, .stop, .Y, .S, .S, .S, .S, .stop, .C, .W, .C, .L, .F, .L, .F]

def mitochondrialData : List Residue := [
  .K, .N, .K, .N, .T, .T, .T, .T, .stop, .S, .stop, .S, .M, .I, .M, .I,
  .Q, .H, .Q, .H, .P, .P, .P, .P, .R, .R, .R, .R, .L, .L, .L, .L,
  .E, .D, .E, .D, .A, .A, .A, .A, .G, .G, .G, .G, .V, .V, .V, .V,
  .stop, .Y, .stop, .Y, .S, .S, .S, .S, .W, .C, .W, .C, .L, .F, .L, .F]

```

```
def selectedData : List Residue := [
  .Q, .N, .Q, .N, .T, .T, .T, .T, .stop, .S, .stop, .S, .M, .I, .M, .I,
  .K, .H, .K, .H, .P, .P, .P, .P, .R, .R, .R, .R, .L, .L, .L, .L,
  .E, .D, .E, .D, .A, .A, .A, .A, .G, .G, .G, .G, .V, .V, .V, .V,
  .stop, .C, .stop, .C, .S, .S, .S, .S, .W, .Y, .W, .Y, .L, .F, .L, .F]

def standard (m : Marker) : Residue := standardData[m.val]!
def mitochondrial (m : Marker) : Residue := mitochondrialData[m.val]!
def selected (m : Marker) : Residue := selectedData[m.val]!
def refinementKey (m : Marker) := (representative m, standard m)
def sameRefinement (m n : Marker) : Prop := refinementKey m = refinementKey n
instance (m n : Marker) : Decidable (sameRefinement m n) := inferInstanceAs (Decidable (_ = _))
def doublet (m n : Marker) : Prop :=
  (m = 11 ∧ n = 52) ∨ (m = 52 ∧ n = 11) ∨
  (m = 9 ∧ n = 54) ∨ (m = 54 ∧ n = 9) ∨
  (m = 31 ∧ n = 62) ∨ (m = 62 ∧ n = 31)
instance (m n : Marker) : Decidable (doublet m n) :=
  inferInstanceAs (Decidable ((_ ∧ _) ∨ (_ ∧ _) ∨ (_ ∧ _) ∨ (_ ∧ _) ∨ (_ ∧ _) ∨ (_ ∧ _)))

theorem code_table_lengths :
    standardData.length = 64 ∧ mitochondrialData.length = 64 ∧ selectedData.length = 64 := by decide
theorem standard_special_fibres :
    (allMarkers.filter (fun m => standard m == .M)) = [14] ∧
    (allMarkers.filter (fun m => standard m == .W)) = [58] ∧
    (allMarkers.filter (fun m => standard m == .stop)) = [48,50,56] := by decide
theorem standard_not_registry_invariant : standard (flip 0) ≠ standard 0 := by decide
theorem no_standard_registry_factorization :
    ¬ ∃ k : Registry → Residue, ∀ m, k (quotientMap m) = standard m := by
  rintro ⟨k,hk⟩
  have same : quotientMap (flip 0) = quotientMap 0 := by decide
  have impossible := (hk (flip 0)).symm.trans ((congrArg k same).trans (hk 0))
  exact standard_not_registry_invariant impossible
theorem sixty_one_refinement_cells :
    ((allMarkers.map refinementKey).eraseDups).length = 61 := by decide
theorem exact_refinement_doublets : ∀ m n : Marker,
    sameRefinement m n ↔ m = n ∨ doublet m n := by decide

-- Any proposed retained-state map must separate distinct refinement keys.
theorem refinement_forced_separation {Z : Type} (z : Marker → Z)
    (u : Z → Marker) (v : Z → Residue)
    (hu : ∀ m, u (z m) = representative m) (hv : ∀ m, v (z m) = standard m)
    {m n : Marker} (neq : refinementKey m ≠ refinementKey n) : z m ≠ z n := by
  intro h
  apply neq
  apply Prod.ext
  · exact (hu m).symm.trans ((congrArg u h).trans (hu n))
  · exact (hv m).symm.trans ((congrArg v h).trans (hv n))


-- An explicit 61-point injection certifies the cardinal lower bound without
-- importing a general finite-cardinality library.
def refinementRepresentatives : List Marker := [
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
  16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
  32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
  48, 49, 50, 51, 53, 55, 56, 57, 58, 59, 60, 61, 63]
def refinementPoint (i : Fin 61) : Marker := refinementRepresentatives[i.val]!

theorem refinement_representatives_distinct : ∀ i j : Fin 61,
    refinementKey (refinementPoint i) = refinementKey (refinementPoint j) → i = j := by decide

theorem refinement_lower_bound_embedding {Z : Type} (z : Marker → Z)
    (u : Z → Marker) (v : Z → Residue)
    (hu : ∀ m, u (z m) = representative m) (hv : ∀ m, v (z m) = standard m) :
    Function.Injective (fun i : Fin 61 => z (refinementPoint i)) := by
  intro i j h
  apply refinement_representatives_distinct i j
  apply Prod.ext
  · exact (hu _).symm.trans ((congrArg u h).trans (hu _))
```

```
  · exact (hv _).symm.trans ((congrArg v h).trans (hv _))

def witnessWC : List (Marker × Marker × Residue) := [
  (0,0,.K),
  (7,7,.T),
  (51,51,.Y),
  (11,11,.S),
  (33,33,.D),
  (25,25,.R),
  (18,18,.Q),
  (21,21,.P),
  (31,31,.L),
  (15,15,.I),
  (61,61,.F),
  (57,57,.C),
  (58,58,.W),
  (14,14,.M),
  (19,19,.H),
  (36,36,.A),
  (43,43,.G),
  (34,34,.E)]

def witnessRY : List (Marker × Marker × Residue) := [
  (0,0,.K),
  (56,44,.V),
  (12,8,.R),
  (11,14,.M),
  (33,49,.Y),
  (38,55,.S),
  (18,19,.H),
  (42,63,.F),
  (1,1,.N),
  (48,32,.E),
  (61,41,.G),
  (24,28,.L),
  (4,4,.T),
  (23,22,.P),
  (49,33,.D),
  (19,18,.Q),
  (9,13,.I),
  (53,37,.A),
  (46,59,.C)]

def witnessMK : List (Marker × Marker × Residue) := [
  (0,42,.G),
  (7,35,.D),
  (12,46,.V),
  (11,39,.A),
  (33,24,.R),
  (25,4,.T),
  (18,9,.S),
  (42,21,.P),
  (62,61,.F),
  (48,58,.W),
  (47,31,.L),
  (39,19,.H),
  (55,51,.Y),
  (5,32,.E),
  (28,14,.M),
  (50,57,.C),
  (36,18,.Q),
  (20,2,.K),
  (29,12,.I),
  (22,1,.N)]

def encodeRY (m : Marker) : Marker :=
  let w := encode m
  let f := fun b : Base => if b = 2 then (3 : Base) else if b = 3 then 2 else b
  decode (f w.1, f w.2.1, f w.2.2)
```

```
def encodeMK (m : Marker) : Marker :=
  let w := encode m
  let f := fun b : Base => if b = 0 then (2 : Base) else if b = 1 then 0 else if b = 2 then 1 else 3
  decode (f w.1, f w.2.1, f w.2.2)
def validWitness (encoding : Marker → Marker) (ws : List (Marker × Marker × Residue)) : Prop :=
  (∀ t ∈ ws, encoding t.1 = t.2.1 ∧ standard t.2.1 = t.2.2 ∧ t.2.2 ≠ .stop) ∧
  (ws.map (fun t => representative t.1)).Nodup ∧
  (ws.map (fun t => t.2.2)).Nodup
instance (e : Marker → Marker) (w : List (Marker × Marker × Residue)) :
    Decidable (validWitness e w) := inferInstanceAs (Decidable (_ ∧ _ ∧ _))

theorem matching_witnesses :
    validWitness id witnessWC ∧ witnessWC.length = 18 ∧
    validWitness encodeRY witnessRY ∧ witnessRY.length = 19 ∧
    validWitness encodeMK witnessMK ∧ witnessMK.length = 20 := by decide

def hallWC : List Marker := [7,11,25,21,6,4,9,10,22]
def neighborsWC : List Residue := [.A,.C,.G,.P,.R,.S,.T]

def hallRY : List Marker := [7,25,21,6,4,5,9,10,22]
def neighborsRY : List Residue := [.A,.F,.I,.L,.P,.S,.T,.V]

def neighborsAt (encoding : Marker → Marker) (rows : List Marker) : List Residue :=
  ((rows.flatMap orbit).map (fun m => standard (encoding m))).eraseDups.filter (fun r => r != .stop)

theorem wc_hall_certificate : hallWC.length = 9 ∧ neighborsWC.length = 7 ∧
    (∀ r : Residue, r ∈ neighborsAt id hallWC ↔ r ∈ neighborsWC) := by
  refine ⟨by decide, by decide, ?_⟩
  intro r; cases r <;> decide
theorem ry_hall_certificate : hallRY.length = 9 ∧ neighborsRY.length = 8 ∧
    (∀ r : Residue, r ∈ neighborsAt encodeRY hallRY ↔ r ∈ neighborsRY) := by
  refine ⟨by decide, by decide, ?_⟩
  intro r; cases r <;> decide

-- F4. Cell routing is a distinct, addressed relation.
def quartet (pre : Nat) : Bool := [13,7,5,6,1,11,9,10].contains pre
def cell (m : Marker) : Nat := 2*(m.val/4) + if quartet (m.val/4) then 0 else m.val%2
def cellRepresentative (c : Nat) : Marker := mkMarker (4*(c/2) + c%2)
def cells := (allMarkers.map cell).eraseDups
def exchange (t : Marker → Residue) (a b : Nat) (m : Marker) : Residue :=
  if cell m = a then t (cellRepresentative b)
  else if cell m = b then t (cellRepresentative a) else t m

def exchangeGuard (a b : Nat) : Prop :=
  a ∈ cells ∧ b ∈ cells ∧ a ≠ b ∧ a ≠ 6 ∧ b ≠ 6 ∧
  (a%2 = b%2) ∧ (quartet (a/2) = quartet (b/2)) ∧
  ((a/8 = b/8 ∧ a/2%4 ≠ b/2%4) ∨ (a/8 ≠ b/8 ∧ a/2%4 = b/2%4)) ∧
  selected (cellRepresentative a) ≠ .stop ∧ selected (cellRepresentative b) ≠ .stop
instance (a b : Nat) : Decidable (exchangeGuard a b) := by unfold exchangeGuard; infer_instance
def afterFirst := exchange selected 25 29
def afterSecond := exchange afterFirst 0 8

theorem cell_partition_counts : cells.length = 24 ∧
    (cells.filter (fun c => (allMarkers.filter (fun m => cell m == c)).length == 4)).length = 8 ∧
    (cells.filter (fun c => (allMarkers.filter (fun m => cell m == c)).length == 2)).length = 16 := by
     decide
theorem selected_repertoire :
    ((allMarkers.map selected).eraseDups.filter (fun r => r != .stop)).length = 20 := by decide
theorem two_exchange_guards : exchangeGuard 25 29 ∧ exchangeGuard 0 8 := by decide
theorem two_exchanges_reach_mitochondrial : ∀ m : Marker, afterSecond m = mitochondrial m := by decide
theorem endpoint_differences :
    (allMarkers.filter (fun m => standard m != mitochondrial m)) = [8,10,12,56] := by decide
theorem initialized_common_tape :
    (∀ t ∈ [selected,afterFirst,afterSecond,standard],
      t 14 = .M ∧ t 39 = .A ∧ t 48 = .stop) := by decide

def endpointSensePairs :
  (((allMarkers.map (fun m => (m,standard m))) ++
    (allMarkers.map (fun m => (m,mitochondrial m)))).eraseDups).filter (fun t => t.2 != .stop)
```

```
theorem sixty_three_endpoint_sense_pairs : endpointSensePairs.length = 63 := by decide

-- F5. A real selection computation on finite branch lists, over exact rationals.
def selectBranches (bs : List (Residue × Rat)) : List Residue :=
  (bs.filter (fun b => decide (0 < b.2) && bs.all (fun c => decide (c.2 ≤ b.2)))).map Prod.fst

def aua (u : Rat) := selectBranches [(.I,1+3*u),(.M,2+u)]
def uga (u : Rat) := selectBranches [(.W,1-2*u)]
def agr (u : Rat) := selectBranches [(.R,2*u-1)]
def readout (u : Rat) (m : Marker) : List Residue :=
  if m = 12 then aua u else if m = 14 then selectBranches [(.M,2+u)]
  else if m = 56 then uga u else if m = 8 ∨ m = 10 then agr u
  else if mitochondrial m = .stop then [] else [mitochondrial m]
def outputList (r : Residue) : List Residue := if r = .stop then [] else [r]

theorem branch_score_difference (u : Rat) : (1+3*u)-(2+u) = 2*u-1 := by grind
theorem aua_positive_branches (u : Rat) (hu : 0 ≤ u) : 0 < 1+3*u ∧ 0 < 2+u := by grind
theorem control_before_wall (u : Rat) (hu : 0 ≤ u) (hw : u < 1/2) :
    aua u = [.M] ∧ uga u = [.W] ∧ agr u = [] := by
  have h1 : 0 < 1+3*u := by grind
  have h2 : 0 < 2+u := by grind
  have h3 : ¬ 2+u ≤ 1+3*u := by grind
  have h4 : 1+3*u ≤ 2+u := by grind
  have h5 : 0 < 1-2*u := by grind
  have h6 : ¬ 0 < 2*u-1 := by grind
  simp [aua,uga,agr,selectBranches,List.filter_nil,Rat.le_refl,h1,h2,h3,h4,h5,h6]
theorem control_at_wall : aua (1/2) = [.I,.M] ∧ uga (1/2) = [] ∧ agr (1/2) = [] := by decide +kernel
theorem control_after_wall (u : Rat) (hw : 1/2 < u) :
    aua u = [.I] ∧ uga u = [] ∧ agr u = [.R] := by
  have h1 : 0 < 1+3*u := by grind
  have h2 : 0 < 2+u := by grind
  have h3 : 2+u ≤ 1+3*u := by grind
  have h4 : ¬ 1+3*u ≤ 2+u := by grind
  have h5 : ¬ 0 < 1-2*u := by grind
  have h6 : 0 < 2*u-1 := by grind
  simp [aua,uga,agr,selectBranches,List.filter_nil,Rat.le_refl,h1,h2,h3,h4,h5,h6]
theorem control_endpoints_all_codons : ∀ m : Marker,
    readout 0 m = outputList (mitochondrial m) ∧
    readout 1 m = outputList (standard m) := by decide +kernel
theorem protected_aug_control (u : Rat) (hu : 0 ≤ u) : readout u 14 = [.M] := by
  have hp : 0 < 2+u := by grind
  simp [readout,selectBranches,List.filter_nil,Rat.le_refl,hp]
theorem common_stop_control (u : Rat) : readout u 48 = [] := by
  have h : mitochondrial 48 = .stop := by decide
  simp [readout,h]

-- The nested-singleton obstruction requires neither continuity nor enumeration.
theorem nested_singleton_obstruction {P : Type} (earlier later : P → Prop)
    (nested : ∀ r, later r → earlier r)
    (singleton : ∀ r s, earlier r → earlier s → r = s)
    (r : P) (present : later r) : ∀ s, earlier s ↔ s = r := by
  intro s; constructor
  · intro hs; exact singleton s r hs (nested r present)
  · intro h; subst s; exact nested r present

theorem decreasing_crossing_scores (t : Rat) (_lo : 0 ≤ t) (hi : t ≤ 1) :
    0 < 3-2*t ∧ 0 < 2-t/2 ∧
    (3-2*t)-(2-t/2) = 1-3*t/2 := by grind
theorem decreasing_crossing_order (t : Rat) :
    (2-t/2 < 3-2*t ↔ t < 2/3) ∧
    (3-2*t < 2-t/2 ↔ 2/3 < t) := by
  have half : t/2+t/2 = t := by grind
  have third : (2/3:Rat)+(2/3)+(2/3) = 2 := by decide +kernel
  grind

-- F6. Directed readout and release truth table.
def incoming (a : Base) : Word := (3,0,a)
def outgoing (a : Base) : Word := (0,3,complement a)
def narrow (a : Base) : Bool := a == 1 || a == 3
```

```
def incomingGate (a : Base) := narrow a
def outgoingGate (a : Base) := narrow (complement (complement a))
def pairedRelease (a : Base) := incomingGate a && outgoingGate a

theorem complement_involution : ∀ a : Base, complement (complement a) = a := by decide
theorem directed_word_relation : ∀ a : Base, outgoing a = wordFlip (incoming a) := by decide
theorem release_exactly_pyrimidine : ∀ a : Base,
    pairedRelease a = true ↔ a = 1 ∨ a = 3 := by decide
theorem untransported_gate_impossible : ∀ a : Base,
    (narrow a && narrow (complement a)) = false := by decide
theorem junction_aug_witness :
    decode (incoming 1) = 49 ∧ decode (outgoing 1) = 14 ∧ pairedRelease 1 = true := by decide
theorem junction_four_rows :
    (List.finRange 4).map (fun a => (decode (incoming a),decode (outgoing a),pairedRelease a)) =
    [(48,15,false),(49,14,true),(50,13,false),(51,12,true)] := by decide
theorem exact_register_transport_unique (transport : Base → Base)
    (h : ∀ a, transport (complement a) = a) : transport = complement := by
  funext b
  have hb := h (complement b)
  simpa only [complement_involution] using hb

-- F7. An explicit sulfur product identity map (heavy atoms, no stereo claim).
inductive Element where
  | carbon | nitrogen | oxygen | sulfur | hydrogen
  deriving DecidableEq, Repr, Inhabited
abbrev Heavy := Fin 9

def bondOrder (edges : List (Nat × Nat × Nat)) (i j : Nat) : Nat :=
  match edges with
  | [] => 0
  | (a,b,k)::rest => if (i = a ∧ j = b) ∨ (i = b ∧ j = a) then k else bondOrder rest i j
-- Source order: 5,4b,10b,s4,c_alpha,c_prime,n_alpha,o_x,o_t.
def sourceMetElements : List Element :=
  [.carbon,.carbon,.carbon,.sulfur,.carbon,.carbon,.nitrogen,.oxygen,.oxygen]
def sourceMetBonds : List (Nat × Nat × Nat) :=
  [(0,4,1),(0,1,1),(1,3,1),(3,2,1),(4,5,1),(4,6,1),(5,7,2),(5,8,1)]
-- Target order: N,C_alpha,C_prime,O_x,O_t,C_beta,C_gamma,S,C_epsilon.
def targetMetElements : List Element :=
  [.nitrogen,.carbon,.carbon,.oxygen,.oxygen,.carbon,.carbon,.sulfur,.carbon]
def targetMetBonds : List (Nat × Nat × Nat) :=
  [(0,1,1),(1,2,1),(2,3,2),(2,4,1),(1,5,1),(5,6,1),(6,7,1),(7,8,1)]
def metMap (v : Heavy) : Fin 9 :=
  ⟨([5,6,8,7,1,2,0,3,4][v.val]! : Nat)%9, Nat.mod_lt _ (by decide)⟩

theorem methionine_atom_map_bijective :
    (∀ a b : Heavy, metMap a = metMap b → a = b) ∧
    (∀ b : Heavy, ∃ a : Heavy, metMap a = b) := by decide
theorem methionine_element_preservation : ∀ a : Heavy,
    sourceMetElements[a.val]! = targetMetElements[(metMap a).val]! := by decide
theorem methionine_bond_preservation : ∀ a b : Heavy,
    bondOrder sourceMetBonds a.val b.val = bondOrder targetMetBonds (metMap a).val (metMap b).val := by
     decide
theorem methionine_root_preservation : metMap 4 = 1 := by decide

-- The small decoder recognizes this rooted template by all heavy-atom data.
def outgoingIdentity : Residue :=
  if (∀ a : Heavy, sourceMetElements[a.val]! = targetMetElements[(metMap a).val]!) ∧
     (∀ a b : Heavy, bondOrder sourceMetBonds a.val b.val =
       bondOrder targetMetBonds (metMap a).val (metMap b).val) ∧ metMap 4 = 1
  then .M else .stop

theorem outgoing_graph_identifies_met : outgoingIdentity = .M := by decide

-- F8. Finite paths and the projected controller safety proof.
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_invariant {X : Type} {step : X → X → Prop}
    (I : X → Prop) (preserves : ∀ x y, step x y → I x → I y)
    {x y : X} (h : Execution step x y) (hx : I x) : I y := by
  induction h with
  | empty => exact hx
  | next h _ ih => exact ih (preserves _ _ h hx)
theorem execution_preserves_projection {X P : Type} (p : X → P)
    (step : X → X → Prop) (fixed : ∀ x y, step x y → p x = p y)
    {x y : X} (h : Execution step x y) : p x = p y := by
  induction h with
  | empty => rfl
  | next h _ ih => exact (fixed _ _ h).trans ih

inductive Mode where
  | idle | ready | formed | primed | active | stopped | released | halted | rejected
  deriving DecidableEq, Repr
structure Reader where
  mode : Mode
  word : List Residue
  archive : List (List Residue)
  phase : Bool
  position : Nat
  deriving DecidableEq, Repr

def initialReader : Reader := ⟨.idle,[],[],false,0⟩
def openMode (m : Mode) : Prop := m = .active ∨ m = .stopped ∨ m = .released
def startsMet (w : List Residue) : Prop := ∃ tail, w = .M :: tail
def readerSafe (s : Reader) : Prop :=
  (∀ w ∈ s.archive, startsMet w) ∧
  (openMode s.mode → startsMet s.word ∧ s.phase = true ∧ s.position % 3 = 0)

-- This is the word/phase/position projection of the declared controller.
-- Graph formation and token freshness remain interfaces. START uses the
-- computed constitutional identity above, not a literal residue or equality premise.
inductive ReaderStep : Reader → Reader → Prop
  | reset (s) (guard : s.mode = .idle ∨ s.mode = .halted ∨ s.mode = .rejected) :
      ReaderStep s ⟨.ready,[],s.archive,false,0⟩
  | form (s) (guard : s.mode = .ready) : ReaderStep s {s with mode := .formed}
  | gate (s) (guard : s.mode = .formed) : ReaderStep s {s with mode := .primed}
  | fail (s) (guard : s.mode = .formed) : ReaderStep s {s with mode := .rejected}
  | start (s) (guard : s.mode = .primed)
      (empty : s.word = []) (unset : s.phase = false) :
      ReaderStep s ⟨.active,[outgoingIdentity],s.archive,true,3⟩
  | extend (s) (product : Residue) (guard : s.mode = .active) :
      ReaderStep s {s with word := s.word ++ [product], position := s.position+3}
  | observe (s) (guard : s.mode = .active) : ReaderStep s {s with mode := .stopped}
  | release (s) (guard : s.mode = .stopped) : ReaderStep s {s with mode := .released}
  | halt (s) (guard : s.mode = .released) :
      ReaderStep s ⟨.halted,[],s.archive ++ [s.word],false,s.position⟩
  | route (s) : ReaderStep s s

theorem starts_met_append (w : List Residue) (r : Residue)
    (h : startsMet w) : startsMet (w ++ [r]) := by
  obtain ⟨tail,ht⟩ := h
  exact ⟨tail ++ [r], by simp [ht]⟩
theorem initial_reader_safe : readerSafe initialReader := by
  simp [readerSafe,initialReader,openMode]
theorem reader_step_preserves (x y : Reader) (step : ReaderStep x y)
    (hx : readerSafe x) : readerSafe y := by
  rcases hx with ⟨archiveSafe,activeSafe⟩
  cases step with
  | reset guard => exact ⟨archiveSafe,by simp [openMode]⟩
  | form guard => exact ⟨archiveSafe,by simp [openMode]⟩
  | gate guard => exact ⟨archiveSafe,by simp [openMode]⟩
  | fail guard => exact ⟨archiveSafe,by simp [openMode]⟩
```

```
  | start guard empty unset =>
      exact ⟨archiveSafe,by simp [startsMet,outgoing_graph_identifies_met]⟩
  | extend product guard =>
      obtain ⟨hw,hp,hn⟩ := activeSafe (Or.inl guard)
      refine ⟨archiveSafe, ?_⟩
      intro _
      exact ⟨starts_met_append _ _ hw,hp,by change (x.position+3)%3 = 0; omega⟩
  | observe guard =>
      exact ⟨archiveSafe,fun _ => activeSafe (Or.inl guard)⟩
  | release guard =>
      exact ⟨archiveSafe,fun _ => activeSafe (Or.inr (Or.inl guard))⟩
  | halt guard =>
      have hs := (activeSafe (Or.inr (Or.inr guard))).1
      constructor
      · intro w hw
        simp only [List.mem_append,List.mem_cons,List.not_mem_nil,or_false] at hw
        rcases hw with h | h
        · exact archiveSafe w h
        · simpa [h] using hs
      · simp [openMode]
  | route => exact ⟨archiveSafe,activeSafe⟩
theorem all_finite_runs_met_first {s : Reader} (run : Execution ReaderStep initialReader s) :
    readerSafe s := execution_invariant readerSafe reader_step_preserves run initial_reader_safe

-- F9. Persistent origins, local edits and monotone non-recycled resources.
abbrev AtomID := Nat × Nat × Nat
structure AtomRecord where
  id : AtomID
  element : Element
  deriving DecidableEq, Repr

def copyAtom (batch : Nat) (a : Nat × Nat) : AtomID := (batch,a.1,a.2)
theorem copy_origin_injective (batch : Nat) : Function.Injective (copyAtom batch) := by
  intro a b h
  cases a; cases b
  simp [copyAtom] at h
  exact Prod.ext h.1 h.2
theorem different_copies_disjoint (a b : Nat) (h : a ≠ b) (v w : Nat × Nat) :
    copyAtom a v ≠ copyAtom b w := by
  intro heq
  exact h (congrArg Prod.fst heq)

structure LocalBond where
  order : Nat
  freeLeft : Nat
  freeRight : Nat
  deriving DecidableEq, Repr

def splitBond (s : LocalBond) : LocalBond := ⟨s.order-1,s.freeLeft+1,s.freeRight+1⟩
def joinBond (s : LocalBond) : LocalBond := ⟨s.order+1,s.freeLeft-1,s.freeRight-1⟩
def endpointCapacities (s : LocalBond) := (s.order+s.freeLeft,s.order+s.freeRight)

theorem split_conserves_endpoint_capacity (s : LocalBond) (h : 0 < s.order) :
    endpointCapacities (splitBond s) = endpointCapacities s := by
  apply Prod.ext <;> simp [endpointCapacities,splitBond] <;> omega
theorem join_conserves_endpoint_capacity (s : LocalBond)
    (hl : 0 < s.freeLeft) (hr : 0 < s.freeRight) :
    endpointCapacities (joinBond s) = endpointCapacities s := by
  apply Prod.ext <;> simp [endpointCapacities,joinBond] <;> omega
theorem join_respects_bond_order_limit (s : LocalBond) (h : s.order < 3) :
    (joinBond s).order ≤ 3 := by simp [joinBond]; omega

def updateAt {A B : Type} [DecidableEq A] (f : A → B) (a : A) (b : B) : A → B :=
  fun x => if x = a then b else f x

theorem disjoint_coordinate_updates_commute {A B : Type} [DecidableEq A]
    (f : A → B) (a c : A) (b d : B) (h : a ≠ c) :
    updateAt (updateAt f a b) c d = updateAt (updateAt f c d) a b := by
  funext x
```

```
  simp only [updateAt]
  grind

def consume (used : Nat → Bool) (token : Nat) : Nat → Bool := updateAt used token true
inductive ResourceStep : (Nat → Bool) → (Nat → Bool) → Prop
  | take (used) (token) (fresh : used token = false) : ResourceStep used (consume used token)

theorem consumption_persists (used : Nat → Bool) (token k : Nat)
    (h : used k = true) : consume used token k = true := by
  simp [consume,updateAt,h]
theorem newly_consumed_unavailable (used : Nat → Bool) (token : Nat) :
    consume used token token = true := by simp [consume,updateAt]
theorem resource_step_monotone (x y : Nat → Bool) (h : ResourceStep x y)
    (k : Nat) (hk : x k = true) : y k = true := by
  cases h with
  | take token fresh => exact consumption_persists x token k hk
theorem resource_run_no_recycling {x y : Nat → Bool} (run : Execution ResourceStep x y)
    (k : Nat) (used : x k = true) : y k = true := by
  exact execution_invariant (fun s => s k = true)
    (fun a b h hk => resource_step_monotone a b h k hk) run used

def totalCost : List Nat → Nat | [] => 0 | n::ns => n+totalCost ns
theorem costs_add (xs ys : List Nat) : totalCost (xs++ys) = totalCost xs+totalCost ys := by
  induction xs with
  | nil => simp [totalCost]
  | cons n ns ih => simp [totalCost,ih,Nat.add_assoc]
theorem prefix_cost_bounded (xs ys : List Nat) (budget : Nat)
    (h : totalCost (xs++ys) ≤ budget) : totalCost xs ≤ budget := by
  rw [costs_add] at h; omega
theorem finite_packet_budget (library packet n : Nat) :
    totalCost (library :: List.replicate n packet) = library+n*packet := by
  induction n with
  | zero => simp [totalCost]
  | succ n ih =>
      simp [totalCost,List.replicate_succ,Nat.succ_mul] at *
      omega

end Article75

-- Per-theorem kernel dependency audit.
#print axioms Article75.encoding_left_inverse
#print axioms Article75.encoding_right_inverse
#print axioms Article75.flip_involution
#print axioms Article75.reverse_involution
#print axioms Article75.generators_commute
#print axioms Article75.flip_has_no_fixed_point
#print axioms Article75.encoding_flip_transport
#print axioms Article75.encoding_reverse_transport
#print axioms Article75.reversal_is_not_ordinary
#print axioms Article75.enumeration_complete
#print axioms Article75.difference_fibres
#print axioms Article75.difference_transport
#print axioms Article75.complete_invariant_six_bits
#print axioms Article75.representative_classification
#print axioms Article75.representative_in_orbit
#print axioms Article75.representative_idempotent
#print axioms Article75.twenty_orbits
#print axioms Article75.orbit_size_dichotomy
#print axioms Article75.orbit_size_distribution
#print axioms Article75.fixed_point_counts
#print axioms Article75.short_orbit_iff_palindrome
#print axioms Article75.invariant_on_orbit
#print axioms Article75.invariant_factors_uniquely
#print axioms Article75.normalized_action
#print axioms Article75.normalized_zero_tags
#print axioms Article75.normalized_representatives_unique
#print axioms Article75.residual_counts
#print axioms Article75.code_table_lengths
#print axioms Article75.standard_special_fibres
```

```
#print axioms Article75.standard_not_registry_invariant
#print axioms Article75.no_standard_registry_factorization
#print axioms Article75.sixty_one_refinement_cells
#print axioms Article75.exact_refinement_doublets
#print axioms Article75.refinement_forced_separation
#print axioms Article75.refinement_representatives_distinct
#print axioms Article75.refinement_lower_bound_embedding
#print axioms Article75.matching_witnesses
#print axioms Article75.wc_hall_certificate
#print axioms Article75.ry_hall_certificate
#print axioms Article75.cell_partition_counts
#print axioms Article75.selected_repertoire
#print axioms Article75.two_exchange_guards
#print axioms Article75.two_exchanges_reach_mitochondrial
#print axioms Article75.endpoint_differences
#print axioms Article75.initialized_common_tape
#print axioms Article75.sixty_three_endpoint_sense_pairs
#print axioms Article75.branch_score_difference
#print axioms Article75.aua_positive_branches
#print axioms Article75.control_before_wall
#print axioms Article75.control_at_wall
#print axioms Article75.control_after_wall
#print axioms Article75.control_endpoints_all_codons
#print axioms Article75.protected_aug_control
#print axioms Article75.common_stop_control
#print axioms Article75.nested_singleton_obstruction
#print axioms Article75.decreasing_crossing_scores
#print axioms Article75.decreasing_crossing_order
#print axioms Article75.complement_involution
#print axioms Article75.directed_word_relation
#print axioms Article75.release_exactly_pyrimidine
#print axioms Article75.untransported_gate_impossible
#print axioms Article75.junction_aug_witness
#print axioms Article75.junction_four_rows
#print axioms Article75.exact_register_transport_unique
#print axioms Article75.methionine_atom_map_bijective
#print axioms Article75.methionine_element_preservation
#print axioms Article75.methionine_bond_preservation
#print axioms Article75.methionine_root_preservation
#print axioms Article75.outgoing_graph_identifies_met
#print axioms Article75.execution_composition
#print axioms Article75.execution_invariant
#print axioms Article75.execution_preserves_projection
#print axioms Article75.starts_met_append
#print axioms Article75.initial_reader_safe
#print axioms Article75.reader_step_preserves
#print axioms Article75.all_finite_runs_met_first
#print axioms Article75.copy_origin_injective
#print axioms Article75.different_copies_disjoint
#print axioms Article75.split_conserves_endpoint_capacity
#print axioms Article75.join_conserves_endpoint_capacity
#print axioms Article75.join_respects_bond_order_limit
#print axioms Article75.disjoint_coordinate_updates_commute
#print axioms Article75.consumption_persists
#print axioms Article75.newly_consumed_unavailable
#print axioms Article75.resource_step_monotone
#print axioms Article75.resource_run_no_recycling
#print axioms Article75.costs_add
#print axioms Article75.prefix_cost_bounded
#print axioms Article75.finite_packet_budget
```

Actual per-theorem dependency audit

The following is the compiler output from the successful build. Every named theorem is listed. Only the standard foundational axioms described above may occur; there are no custom theorem axioms

or proof holes.

```
'Article75.encoding_left_inverse' does not depend on any axioms
'Article75.encoding_right_inverse' does not depend on any axioms
'Article75.flip_involution' depends on axioms: [propext, Quot.sound]
'Article75.reverse_involution' does not depend on any axioms
'Article75.generators_commute' depends on axioms: [propext, Quot.sound]
'Article75.flip_has_no_fixed_point' depends on axioms: [propext, Quot.sound]
'Article75.encoding_flip_transport' depends on axioms: [propext, Quot.sound]
'Article75.encoding_reverse_transport' does not depend on any axioms
'Article75.reversal_is_not_ordinary' depends on axioms: [propext]
'Article75.enumeration_complete' depends on axioms: [propext]
'Article75.difference_fibres' depends on axioms: [propext, Quot.sound]
'Article75.difference_transport' depends on axioms: [propext, Quot.sound]
'Article75.complete_invariant_six_bits' depends on axioms: [propext, Quot.sound]
'Article75.representative_classification' depends on axioms: [propext, Quot.sound]
'Article75.representative_in_orbit' depends on axioms: [propext, Quot.sound]
'Article75.representative_idempotent' depends on axioms: [propext, Quot.sound]
'Article75.twenty_orbits' depends on axioms: [propext, Quot.sound]
'Article75.orbit_size_dichotomy' depends on axioms: [propext, Quot.sound]
'Article75.orbit_size_distribution' depends on axioms: [propext, Quot.sound]
'Article75.fixed_point_counts' depends on axioms: [propext, Quot.sound]
'Article75.short_orbit_iff_palindrome' depends on axioms: [propext, Quot.sound]
'Article75.invariant_on_orbit' depends on axioms: [propext, Quot.sound]
'Article75.invariant_factors_uniquely' depends on axioms: [propext, Quot.sound]
'Article75.normalized_action' depends on axioms: [propext, Quot.sound]
'Article75.normalized_zero_tags' depends on axioms: [propext, Quot.sound]
'Article75.normalized_representatives_unique' depends on axioms: [propext, Quot.sound]
'Article75.residual_counts' depends on axioms: [propext, Quot.sound]
'Article75.code_table_lengths' does not depend on any axioms
'Article75.standard_special_fibres' depends on axioms: [propext]
'Article75.standard_not_registry_invariant' depends on axioms: [propext, Quot.sound]
'Article75.no_standard_registry_factorization' depends on axioms: [propext, Quot.sound]
'Article75.sixty_one_refinement_cells' depends on axioms: [propext, Quot.sound]
'Article75.exact_refinement_doublets' depends on axioms: [propext, Quot.sound]
'Article75.refinement_forced_separation' depends on axioms: [propext, Quot.sound]
'Article75.refinement_representatives_distinct' depends on axioms: [propext, Quot.sound]
'Article75.refinement_lower_bound_embedding' depends on axioms: [propext, Quot.sound]
'Article75.matching_witnesses' depends on axioms: [propext, Quot.sound]
'Article75.wc_hall_certificate' depends on axioms: [propext, Quot.sound]
'Article75.ry_hall_certificate' depends on axioms: [propext, Quot.sound]
'Article75.cell_partition_counts' does not depend on any axioms
'Article75.selected_repertoire' does not depend on any axioms
'Article75.two_exchange_guards' depends on axioms: [propext, Quot.sound]
'Article75.two_exchanges_reach_mitochondrial' does not depend on any axioms
'Article75.endpoint_differences' depends on axioms: [propext]
'Article75.initialized_common_tape' depends on axioms: [propext]
'Article75.sixty_three_endpoint_sense_pairs' does not depend on any axioms
'Article75.branch_score_difference' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.aua_positive_branches' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.control_before_wall' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.control_at_wall' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.control_after_wall' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.control_endpoints_all_codons' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.protected_aug_control' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.common_stop_control' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.nested_singleton_obstruction' does not depend on any axioms
'Article75.decreasing_crossing_scores' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.decreasing_crossing_order' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.complement_involution' depends on axioms: [propext, Quot.sound]
'Article75.directed_word_relation' depends on axioms: [propext, Quot.sound]
'Article75.release_exactly_pyrimidine' depends on axioms: [propext, Quot.sound]
'Article75.untransported_gate_impossible' depends on axioms: [propext, Quot.sound]
'Article75.junction_aug_witness' depends on axioms: [propext, Quot.sound]
'Article75.junction_four_rows' depends on axioms: [propext, Quot.sound]
'Article75.exact_register_transport_unique' depends on axioms: [propext, Quot.sound]
'Article75.methionine_atom_map_bijective' depends on axioms: [propext, Quot.sound]
'Article75.methionine_element_preservation' does not depend on any axioms
'Article75.methionine_bond_preservation' does not depend on any axioms
'Article75.methionine_root_preservation' depends on axioms: [propext]
```

```
'Article75.outgoing_graph_identifies_met' depends on axioms: [propext]
'Article75.execution_composition' does not depend on any axioms
'Article75.execution_invariant' does not depend on any axioms
'Article75.execution_preserves_projection' does not depend on any axioms
'Article75.starts_met_append' depends on axioms: [propext]
'Article75.initial_reader_safe' depends on axioms: [propext, Quot.sound]
'Article75.reader_step_preserves' depends on axioms: [propext, Quot.sound]
'Article75.all_finite_runs_met_first' depends on axioms: [propext, Quot.sound]
'Article75.copy_origin_injective' depends on axioms: [propext]
'Article75.different_copies_disjoint' does not depend on any axioms
'Article75.split_conserves_endpoint_capacity' depends on axioms: [propext, Quot.sound]
'Article75.join_conserves_endpoint_capacity' depends on axioms: [propext, Quot.sound]
'Article75.join_respects_bond_order_limit' depends on axioms: [propext, Quot.sound]
'Article75.disjoint_coordinate_updates_commute' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article75.consumption_persists' depends on axioms: [propext]
'Article75.newly_consumed_unavailable' depends on axioms: [propext]
'Article75.resource_step_monotone' depends on axioms: [propext]
'Article75.resource_run_no_recycling' depends on axioms: [propext]
'Article75.costs_add' depends on axioms: [propext]
'Article75.prefix_cost_bounded' depends on axioms: [propext, Quot.sound]
'Article75.finite_packet_budget' depends on axioms: [propext, Quot.sound]
```

Publication description

Supported description: “A partial finite/rational Lean 4 formalization of Article 75, containing 89 kernel-checked theorems, complete source, dependency audit and an explicit coverage map.” This description is deliberately narrower than a full-paper or molecular-realization certificate.