Formal Proof Dossier

Article 77: Carbon traceable protocell construction from substituted chrysene stocks

Charles D. Schaper, Ph.D.

Annals of the Chrysene Formalism, Vol. 1, No. 4
Manuscript pages 1567–1605

Verification supplement, 25 October 2026

Kernel-checked partial formalization

Proof source	`Article77.lean`
Tested toolchain	Lean 4.24.0
Imports	`Std`, `Init.Data.Rat.Lemmas`
Named theorems	**125**
Per-theorem dependency audits	**125 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 formal proof of the entire manuscript. It proves universal finite-execution and provenance invariants in an explicit typed projection, exact integer production and allocation results, a concrete stock-reservation certificate, and universal rational algebra and inequalities supporting the operating example. It does **not** prove real differential-equation existence or threshold crossing, chemical synthesis, a complete molecular lift, stochastic kinetics, or physical division.

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 below is *byte-for-byte identical* to the separately compiled file. From this directory:

```
lean Article77.lean
python3 audit.py /path/to/lean
```

The audit rebuilds the source, checks all 125 transitive dependency reports, rejects proof holes and custom axioms, checks the complete coverage list, and verifies the exact embedded source when this document is present. It writes `build.log` and `audit_summary.json`. The source uses proof-producing tactics, including `omega`, `grind`, `decide` and `decide +kernel`. Rational closed computations use kernel reduction, not `native_decide`. The Python helper is not part of the proof's trusted kernel.

Overleaf: upload this single `Article77_Formal_Proof.tex` file and select pdfLaTeX. It requires no other source, bibliography, image, network access or shell escape. Overleaf typesets the dossier; it does *not* run Lean. Reproduce the proof with the raw UTF-8 `Article77.lean`, not with text extracted from the PDF.

Compiled source SHA-256:
`43c63bbbeaf119ba9ca246bb29dec12af9d92ab53744d74b5031266ef13c811c`

The dossier and source are independently reproducible after relocation. The manuscript and its reference archives are not runtime inputs to the Lean compiler or audit. `input_manifest.json` records the manuscript source snapshot consulted; `theorem_coverage.json` maps every named theorem to its group and manuscript anchors. The separate executable verification supplement covers additional graph-certificate and numerical checks that are not silently treated as Lean theorems here.

Representation and assumptions

Origin and ownership projection. An origin is a triple of natural numbers naming admission, constituent and atom. An admitted list is required to have no duplicates. A total ownership function assigns each origin one of free, bound, residual, recycled, exterior and inaccessible. Only origins in the admitted list represent present carbon. Fresh admission inserts one previously absent name; internal moves change an ownership coordinate without changing any name. Preparing a nonempty, duplicate-free support requires every origin to be present and free. It binds that support; activation and release are separate typed phases. Release archives the same support, returns it to the free owner class and clears the candidate. Rearming is required before another preparation. All finite runs preserve distinct admitted origins and traced, nonempty archived products.

This machine is an intentionally limited material/phase projection. It does not construct covalent bonds, distinguish every possible molecule, validate eligible precursor chemistry, or assert that all physical histories implement its constructors. Ownership exclusivity follows from a single-valued function, not from a derived chemical sorting mechanism. Archive entries are historical events: an origin may appear again after an explicitly allowed later preparation; the proof does not confuse historical reuse with simultaneous double ownership.

Concrete reservation. The five formula types are the explicit neutral catalogue data. The finite fixture has 38 separately named constituents: $2D_m$, $9T_0$, $23D_2$, $3T_2$ and I_2. All 810 carbon origins are enumerated, their distinctness is checked, and the four role supports are proved disjoint with sizes $40, 162, 168, 440$. This checks origin allocation and formula arithmetic; it does not rebuild the full atom-and-bond graphs or turn the twenty local carbon budgets into prepared aminoacyl modules.

Open balance. The integer ledger has inside, fresh-inside, old-origin inward and outward counters. Every finite list of boundary events satisfies

$$C_{\mathrm{in}} = C_{\mathrm{in}}(0) + I + Q_{\mathrm{old,in}} - Q_{\mathrm{out}}.$$

New stock admitted outside leaves the inside count unchanged. Export followed by return increments both crossing counters but not the fresh-inside counter. This is an exact signed algebraic ledger. Availability, nonnegative physical counts and correspondence between event amounts and the labelled machine are separate obligations; the file does not assert a formal refinement between these two projections.

Integer operation. Counts are natural numbers. A production block consumes ten donors and

two carriers only when both guards hold; it produces one retained four-residue chain, one lumen four-residue chain, two free residues, ten remnants and two spent carriers. Deactivation is enabled only when an active chain exists. Both steps preserve the three ledger rows and every carbon-weighted combination. Moreover, for every $M, B, W \in \mathbb{N}$ with $B \leq 13M$ and $W \leq B$, the exact state

$$b = 130M - 10B, \quad c = 30M - 2B, \quad p = B - W, \quad w = W,$$
$$l = B, \quad h = z = 2B, \quad r = 10B$$

is reached by a constructive finite execution. This is a genuine universal reachability statement, not just a check of one simulation or an assumed terminal state. Its ordering executes production before deactivation; it does not prove the manuscript's continuous-time floor construction or the uniformly timed approximation.

Coverage mapped to manuscript statements

Source labels below identify manuscript statements, not external LaTeX references. Formal coverage is the exact result described, not every assertion made under an associated label.

Group and anchors	Kernel-checked result	Boundary
F1: `arc:composition`	Finite execution composition and preservation of any invariant proved for each step.	A generic transition theorem; complete molecular interface compatibility is not established by it.
F2: `mat:thm-origin`, `mat:thm-disjoint`, `mat:prop-formula`, `mat:prop-witness`	Injective origin copying; distinct admissions; exclusive/exhaustive ownership; framed updates; typed preparation/release safety for all finite runs; never-deleted old origins; a counterexample to disjoint-write-only guard reasoning; five formula values; the complete 810-origin reservation and disjoint role supports.	A projected ledger and exact finite fixture, not a full covalent graph semantics, molecular eligibility proof, complete product grammar or physical assembly.
F3: `mat:eq-open`, `dyn:carbonprojection`	Universal signed integer boundary balance, unchanged inside count for new outside stock, and separate export/return counters.	No real differentiation, atom-to-event refinement or physically enabled transport theorem is supplied.
F4: `mat:eq-block`, `mat:eq-operatingcarbon`, `mat:thm-countpath`, `dyn:invariants`	Guarded block/deactivation preservation, all-run carbon conservation, exact arbitrary event-count states, carrier floor, event-count bounds, and constructive attainability of every funded integer state.	The continuous functions, floor operations, event threshold times, trajectory error estimates and chemical lift are not formalized in this group.

Group and anchors	Kernel-checked result	Boundary
F5: `dyn:renewal`, `div:conditional`	Prefix and total withdrawal bounds, reserve floors under uniformly bounded finite costs, exact binary-parent count $2^N - 1$, necessary and sufficient independent integer daughter minima, one-pore impossibility and carbon-weighted partition.	No fixed-point existence, physical fission, coupled machinery sorting, or complete daughter viability certificate is established.
F6: `arc:duplex`	Reverse-complement length preservation and involution for every finite RNA word.	No strand geometry, base-pair energetics, decoding or replication.
F7: `dyn:witness`, `dyn:margin`, `dyn:ablation`, `dyn:robust`, `div:geometry`, `div:return`, `phys:transfer`	Universal rational operating algebra and bounds, strict saturation monotonicity, interval-wide positive-speed bounds, exact nominal/robust constants, resource and energy margins, rational boundary screens, count-error algebra and selected allocation/return-map identities.	These results support, but do not prove, real ODE existence/uniqueness, integration-to-hitting, all-time invariant tubes, real square roots/exponentials, division geometry, physical observation or attraction of the complete return map.

Exact operating constants and their logical role

The nominal availability lower bound and threshold speed are

$$\mu = \frac{2850}{3131}, \qquad \delta = \frac{3147}{6262}, \qquad \frac{3/2}{\delta} = \frac{3131}{1049} < \frac{597}{200}.$$

The stronger 8/5 threshold uses lower speed 87626/203515 and the quotient $162812/43813 < 3717/1000$. The robust lower availability and speed are

$$\widehat{\mu} = \frac{717500}{799663}, \qquad \widehat{\delta} = \frac{70245111}{159932600}, \qquad \frac{3/2}{\widehat{\delta}} = \frac{239898900}{70245111} < \frac{427}{125}.$$

These are exact rational values. The source proves interval-wide inequalities rather than merely evaluating endpoints. In particular, factorization and nonnegative-product reasoning establish positive lower speed throughout the stated rational peptide interval. Applying that lower speed to a real solution requires the separate integration and first-hit argument in the manuscript. Accordingly, a checked time quotient here is *not* advertised as a Lean proof that an ODE trajectory hits the threshold.

Likewise, the rational water screens are conditional bracket estimates. A conditional rationalized Donnan inequality is proved from its root equation and lower-root bound. Existence and uniqueness of the real square root and its identification with the physical closure are not supplied by that algebra. The geometric numerical screens do not establish local lumen width, a membrane phase or a neck path. A supply or service bound records stipulated costs, not measured reaction energetics.

Important qualifications and remaining proof obligations

1. **No whole conclusion is imported as an axiom.** The transition constructors, stock formula data and rational coefficients are explicit definitions. Induction proves the run invariants; arithmetic and finite computations prove their stated claims. Generic theorems expose local preservation or bounded-cost hypotheses. No theorem assumes the full Article 77 result as a premise.
2. **Standard foundations remain visible.** The permitted transitive axioms are `propext`, `Classical.choice` and `Quot.sound`. "No custom axioms" is not a claim that every proof is free of Lean's standard foundations. The complete actual dependency report follows the source.
3. **The displayed models are separate projections.** A labelled ownership path, a signed crossing ledger, a natural stoichiometric path and rational algebra need not automatically arise from one common molecular history. No unproved correspondence between them is silently added.
4. **Real analysis remains outside this file.** Carathéodory existence and uniqueness, positivity and compact continuation, integration, concavity and differential comparisons, prescribed switching, floor-threshold path timing, Gronwall estimates, real geometry, derivative norms and fixed-point arguments are not certified here unless reduced to the specific rational algebra explicitly stated.
5. **No chemical or probabilistic realization.** The file does not replay the twenty amino-acid graph certificates, prove their full prepared contexts, derive reaction propensities, supply stereochemistry, establish a pore conductance or membrane phase, or exhibit the shared physical parameters and molecular lift. It also supplies no probability or stochastic-process theorem.
6. **Division and reproduction remain conditional.** Integer allocation and cost identities are not a division mechanism. Neither a self-map nor attraction for a complete funded return cycle follows from a scalar derivative inequality alone. The manuscript's physical neck, complete inheritance and return-section obligations remain open.

Complete Lean 4 source

All definitions, 125 named theorems and 125 dependency-audit commands follow. Typographic Unicode substitutions do not change the actual UTF-8 source embedded in this standalone file.

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

/-!
Article 77: a partial finite/integer/rational formalization.
Carbon traceable protocell construction from substituted chrysene stocks.
The labelled ownership machine below is an explicitly defined projection,
not a verified implementation of molecular bond transformations. Real ODE
existence, probability, molecular synthesis and division paths are not proved.
-/
namespace Article77
set_option maxRecDepth 1000000
set_option maxHeartbeats 0

-- F1. Composable finite executions and invariant transport.
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)
```

```
25
26 theorem execution_invariant {X : Type} {step : X → X → Prop}
27     (I : X → Prop) (preserves : ∀ x y, step x y → I x → I y)
28     {x y : X} (run : Execution step x y) (hx : I x) : I y := by
29   induction run with
30   | empty => exact hx
31   | next h _ ih => exact ih (preserves _ _ h hx)
32
33 -- F2. Typed immutable origin names and a total ownership partition.
34 structure Origin where
35   admission : Nat
36   constituent : Nat
37   atom : Nat
38   deriving DecidableEq, Repr
39
40 inductive Owner where
41   | free | bound | residual | recycled | exterior | inaccessible
42   deriving DecidableEq, Repr
43
44 inductive Phase where
45   | allocation | preparation | production | released
46   deriving DecidableEq, Repr
47
48 def copiedOrigin (batch : Nat) (localName : Nat × Nat) : Origin :=
49   ⟨batch,localName.1,localName.2⟩
50
51 theorem copy_prefix_injective (batch : Nat) : Function.Injective (copiedOrigin batch) := by
52   intro x y h
53   cases x; cases y
54   simp [copiedOrigin,Origin.mk.injEq] at h
55   exact Prod.ext h.1 h.2
56
57 theorem different_admissions_disjoint (a b : Nat) (different : a ≠ b)
58     (x y : Nat × Nat) : copiedOrigin a x ≠ copiedOrigin b y := by
59   intro h
60   exact different (congrArg Origin.admission h)
61
62 structure MaterialState where
63   admitted : List Origin
64   ownership : Origin → Owner
65   phase : Phase
66   candidate : List Origin
67   archive : List (List Origin)
68
69 def updateOwner (f : Origin → Owner) (a : Origin) (o : Owner) : Origin → Owner :=
70   fun x => if x = a then o else f x
71
72 def moveSupport (f : Origin → Owner) (support : List Origin) (o : Owner) : Origin → Owner :=
73   fun x => if x ∈ support then o else f x
74
75 def initialMaterial (origins : List Origin) : MaterialState :=
76   ⟨origins,fun _ => .free,.allocation,[],[]⟩
77
78 def materialSafe (s : MaterialState) : Prop :=
79   s.admitted.Nodup ∧
80   (∀ a ∈ s.candidate, a ∈ s.admitted) ∧
81   (∀ product ∈ s.archive, product.Nodup ∧ ∀ a ∈ product, a ∈ s.admitted)
82
83 -- Admission appends a genuinely fresh origin. A local move changes only an
84 -- ownership coordinate. Preparation requires nonempty distinct free origins;
85 -- release is enabled only after preparation and archives that same support.
86 -- These are material/phase guards, not a chemical reaction model.
87 inductive MaterialStep : MaterialState → MaterialState → Prop
88   | freshAdmission (s) (a : Origin) (fresh : a ∉ s.admitted) :
89       MaterialStep s {s with admitted := a :: s.admitted, ownership := updateOwner s.ownership a .free}
90   | move (s) (a : Origin) (o : Owner) (present : a ∈ s.admitted)
91       (notCandidate : a ∉ s.candidate) :
92       MaterialStep s {s with ownership := updateOwner s.ownership a o}
93   | prepare (s) (support : List Origin) (phase : s.phase = .allocation)
```

```
      (nonempty : support ≠ []) (unique : support.Nodup)
      (available : ∀ a ∈ support, a ∈ s.admitted ∧ s.ownership a = .free) :
      MaterialStep s {s with phase := .preparation, candidate := support, ownership := moveSupport s.
    ownership support .bound}
  | activate (s) (phase : s.phase = .preparation) :
      MaterialStep s {s with phase := .production}
  | release (s) (phase : s.phase = .production)
      (unique : s.candidate.Nodup)
      (bound : ∀ a ∈ s.candidate, s.ownership a = .bound) :
      MaterialStep s {s with phase := .released, candidate := [], ownership := moveSupport s.ownership s.
    candidate .free, archive := s.archive ++ [s.candidate]}
  | rearm (s) (phase : s.phase = .released) :
      MaterialStep s {s with phase := .allocation}

theorem owner_partition_exhaustive (s : MaterialState) (a : Origin) :
    s.ownership a = .free ∨ s.ownership a = .bound ∨
    s.ownership a = .residual ∨ s.ownership a = .recycled ∨
    s.ownership a = .exterior ∨ s.ownership a = .inaccessible := by
  cases s.ownership a <;> simp

theorem owner_partition_disjoint (s : MaterialState) (a : Origin)
    (x y : Owner) (hx : s.ownership a = x) (hy : s.ownership a = y) : x = y := by
  exact hx.symm.trans hy

theorem moved_support_owns (f : Origin → Owner) (support : List Origin)
    (o : Owner) (a : Origin) (ha : a ∈ support) : moveSupport f support o a = o := by
  simp [moveSupport,ha]

theorem moved_support_frame (f : Origin → Owner) (support : List Origin)
    (o : Owner) (a : Origin) (ha : a ∉ support) : moveSupport f support o a = f a := by
  simp [moveSupport,ha]

theorem different_owner_updates_commute (f : Origin → Owner)
    (a b : Origin) (x y : Owner) (different : a ≠ b) :
    updateOwner (updateOwner f a x) b y = updateOwner (updateOwner f b y) a x := by
  funext c
  simp only [updateOwner]
  grind

theorem initial_material_safe (origins : List Origin) (unique : origins.Nodup) :
    materialSafe (initialMaterial origins) := by
  simp [materialSafe,initialMaterial,unique]

theorem material_step_preserves (s t : MaterialState) (step : MaterialStep s t)
    (safe : materialSafe s) : materialSafe t := by
  rcases safe with ⟨admittedUnique,candidateSafe,archiveSafe⟩
  cases step with
  | freshAdmission a fresh =>
      refine ⟨by simp [fresh,admittedUnique],?_,?_⟩
      · intro x hx
        exact List.mem_cons_of_mem a (candidateSafe x hx)
      · intro p hp
        obtain ⟨hu,hm⟩ := archiveSafe p hp
        exact ⟨hu,fun x hx => List.mem_cons_of_mem a (hm x hx)⟩
  | move a o present notCandidate => exact ⟨admittedUnique,candidateSafe,archiveSafe⟩
  | prepare support phase nonempty unique available =>
      exact ⟨admittedUnique,fun a ha => (available a ha).1,archiveSafe⟩
  | activate phase => exact ⟨admittedUnique,candidateSafe,archiveSafe⟩
  | release phase unique bound =>
      refine ⟨admittedUnique,by simp,?_⟩
      intro p hp
      simp only [List.mem_append,List.mem_cons,List.not_mem_nil,or_false] at hp
      rcases hp with old | new
      · exact archiveSafe p old
      · subst p
        exact ⟨unique,candidateSafe⟩
  | rearm phase => exact ⟨admittedUnique,candidateSafe,archiveSafe⟩

theorem all_material_runs_preserve_ancestry (origins : List Origin)
```

```
    (unique : origins.Nodup) {s : MaterialState}
    (run : Execution MaterialStep (initialMaterial origins) s) : materialSafe s := by
  exact execution_invariant materialSafe material_step_preserves run
    (initial_material_safe origins unique)

theorem released_product_origins_unique (origins : List Origin)
    (unique : origins.Nodup) {s : MaterialState}
    (run : Execution MaterialStep (initialMaterial origins) s)
    (product : List Origin) (released : product ∈ s.archive) :
    product.Nodup ∧ ∀ a ∈ product, a ∈ s.admitted := by
  exact (all_material_runs_preserve_ancestry origins unique run).2.2 product released

theorem material_admitted_monotone (s t : MaterialState)
    (step : MaterialStep s t) : ∀ a ∈ s.admitted, a ∈ t.admitted := by
  cases step <;> simp_all

theorem old_origins_never_deleted {s t : MaterialState}
    (run : Execution MaterialStep s t) (a : Origin) (present : a ∈ s.admitted) :
    a ∈ t.admitted := by
  exact execution_invariant (fun u => a ∈ u.admitted)
    (fun x y h hx => material_admitted_monotone x y h a hx) run present

-- These additional invariants distinguish reachable release states from
-- arbitrary tuples with an empty or unavailable candidate.
def readyInvariant (s : MaterialState) : Prop :=
  (s.phase = .preparation ∨ s.phase = .production) →
    s.candidate ≠ [] ∧ s.candidate.Nodup ∧
    ∀ a ∈ s.candidate, s.ownership a = .bound

def typedMaterialSafe (s : MaterialState) : Prop :=
  materialSafe s ∧ readyInvariant s ∧ ∀ p ∈ s.archive, p ≠ []

theorem initial_typed_material_safe (origins : List Origin) (unique : origins.Nodup) :
    typedMaterialSafe (initialMaterial origins) := by
  refine ⟨initial_material_safe origins unique,?_,?_⟩
  · simp [readyInvariant,initialMaterial]
  · simp [initialMaterial]

theorem typed_material_step_preserves (s t : MaterialState)
    (step : MaterialStep s t) (safe : typedMaterialSafe s) : typedMaterialSafe t := by
  rcases safe with ⟨material,ready,nonemptyArchive⟩
  have preserved := material_step_preserves s t step material
  refine ⟨preserved,?_,?_⟩
  · cases step with
    | freshAdmission a fresh =>
        intro phase
        obtain ⟨hne,hu,hb⟩ := ready phase
        refine ⟨hne,hu,?_⟩
        intro x hx
        have different : x ≠ a := by
          intro eq
          subst x
          exact fresh (material.2.1 a hx)
        simp [updateOwner,different,hb x hx]
    | move a o present notCandidate =>
        intro phase
        obtain ⟨hne,hu,hb⟩ := ready phase
        refine ⟨hne,hu,?_⟩
        intro x hx
        have different : x ≠ a := by
          intro eq
          subst x
          exact notCandidate hx
        simp [updateOwner,different,hb x hx]
    | prepare support phase nonempty unique available =>
        intro _
        exact ⟨nonempty,unique,fun a ha => moved_support_owns s.ownership support .bound a ha⟩
    | activate phase =>
        intro _
```

```
        exact ready (Or.inl phase)
    | release phase unique bound => simp [readyInvariant]
    | rearm phase => simp [readyInvariant]
  · cases step with
    | freshAdmission a fresh => exact nonemptyArchive
    | move a o present notCandidate => exact nonemptyArchive
    | prepare support phase nonempty unique available => exact nonemptyArchive
    | activate phase => exact nonemptyArchive
    | release phase unique bound =>
        intro p hp
        simp only [List.mem_append,List.mem_cons,List.not_mem_nil,or_false] at hp
        rcases hp with old | new
        · exact nonemptyArchive p old
        · subst p
          exact (ready (Or.inr phase)).1
    | rearm phase => exact nonemptyArchive

theorem every_release_is_nonempty_and_traced (origins : List Origin)
    (unique : origins.Nodup) {s : MaterialState}
    (run : Execution MaterialStep (initialMaterial origins) s)
    (p : List Origin) (archived : p ∈ s.archive) :
    p ≠ [] ∧ p.Nodup ∧ ∀ a ∈ p, a ∈ s.admitted := by
  have safe := execution_invariant typedMaterialSafe typed_material_step_preserves run
    (initial_typed_material_safe origins unique)
  exact ⟨safe.2.2 p archived,safe.1.2.2 p archived⟩

def releaseEnabled (s : MaterialState) : Prop := s.phase = .production

theorem allocation_cannot_release (s : MaterialState) (phase : s.phase = .allocation) :
    ¬ releaseEnabled s := by simp [releaseEnabled,phase]

theorem released_cannot_release_again (s : MaterialState) (phase : s.phase = .released) :
    ¬ releaseEnabled s := by simp [releaseEnabled,phase]

-- Disjoint writes alone do not preserve an unrelated read guard.
def guardedSecondWrite (s : Bool × Bool) : Option (Bool × Bool) :=
  if s.1 = false then some (s.1,true) else none

def firstWrite (s : Bool × Bool) : Bool × Bool := (true,s.2)

theorem disjoint_writes_can_disable_guard :
    guardedSecondWrite (false,false) = some (false,true) ∧
    guardedSecondWrite (firstWrite (false,false)) = none := by decide

-- The manuscript's 38-constituent reservation fixture is also instantiated.
-- Carbon formula data are explicit; no covalent graph is inferred from them.
inductive Stock where
  | D2 | I2 | T2 | T0 | Dm
  deriving DecidableEq, Repr

def armNumbers : Stock → Nat × Nat
  | .D2 => (0,2)
  | .I2 => (0,3)
  | .T2 => (0,4)
  | .T0 => (4,0)
  | .Dm => (1,1)

def precursorFormula (stock : Stock) : Nat × Nat × Nat :=
  let arms := armNumbers stock
  (18+2*arms.2,12+4*arms.2,arms.1+2*arms.2)

theorem five_neutral_precursor_formulas :
    precursorFormula .D2 = (22,20,4) ∧ precursorFormula .I2 = (24,24,6) ∧
    precursorFormula .T2 = (26,28,8) ∧ precursorFormula .T0 = (18,12,4) ∧
    precursorFormula .Dm = (20,16,3) := by decide

theorem substitution_hydrogen_balance (nh ne : Nat) (sites : nh+ne ≤ 12) :
    12-(nh+ne)+nh+5*ne = 12+4*ne := by omega

```

```
inductive Role where
  | membrane | pore | array | module
  deriving DecidableEq, Repr

def stockReservation : List (Nat × Stock × Role) :=
  ((List.range 2).map (fun i => (i,Stock.Dm,Role.membrane))) ++
  ((List.range 9).map (fun i => (2+i,Stock.T0,Role.pore))) ++
  ((List.range 3).map (fun i => (11+i,Stock.D2,Role.array))) ++
  ((List.range 3).map (fun i => (14+i,Stock.T2,Role.array))) ++
  [(17,Stock.I2,Role.array)] ++
  ((List.range 20).map (fun i => (18+i,Stock.D2,Role.module)))

def reservedOrigins (r : Nat × Stock × Role) : List Origin :=
  (List.range (precursorFormula r.2.1).1).map (fun a => ⟨0,r.1,a⟩)

def allReservedOrigins : List Origin := stockReservation.flatMap reservedOrigins

def roleOrigins (role : Role) : List Origin :=
  (stockReservation.filter (fun r => r.2.2 == role)).flatMap reservedOrigins

theorem reservation_has_38_constituents : stockReservation.length = 38 := by decide

theorem each_reserved_support_unique (r : Nat × Stock × Role) : (reservedOrigins r).Nodup := by
  change ((List.range (precursorFormula r.2.1).1).map
    (fun a => Origin.mk 0 r.1 a)).Pairwise (fun a b => a ≠ b)
  apply List.pairwise_map.mpr
  have unique : (List.range (precursorFormula r.2.1).1).Pairwise (fun a b => a ≠ b) :=
    List.nodup_range
  apply unique.imp
  intro a b different equal
  exact different (congrArg Origin.atom equal)

theorem reserved_origin_parent (r : Nat × Stock × Role) (a : Origin)
    (present : a ∈ reservedOrigins r) : a.constituent = r.1 := by
  simp only [reservedOrigins,List.mem_map] at present
  obtain ⟨n,_,equal⟩ := present
  subst a
  rfl

theorem reservation_parent_names_distinct :
    stockReservation.Pairwise (fun a b => a.1 ≠ b.1) := by decide

theorem reservation_has_810_distinct_carbons :
    allReservedOrigins.length = 810 ∧ allReservedOrigins.Nodup := by
  constructor
  · decide
  · apply List.pairwise_flatMap.mpr
    refine ⟨fun r _ => each_reserved_support_unique r,?_⟩
    apply reservation_parent_names_distinct.imp
    intro a b different x hx y hy equal
    have hx' := reserved_origin_parent a x hx
    have hy' := reserved_origin_parent b y hy
    exact different (hx'.symm.trans ((congrArg Origin.constituent equal).trans hy'))

theorem reservation_role_counts :
    (roleOrigins .membrane).length = 40 ∧ (roleOrigins .pore).length = 162 ∧
    (roleOrigins .array).length = 168 ∧ (roleOrigins .module).length = 440 := by decide

def reservedParentRole (parent : Nat) : Role :=
  if parent < 2 then .membrane else if parent < 11 then .pore
  else if parent < 18 then .array else .module

theorem reservation_parent_role_correct : ∀ r ∈ stockReservation,
    reservedParentRole r.1 = r.2.2 := by decide

theorem role_origin_has_unique_parent_role (role : Role) (x : Origin)
    (present : x ∈ roleOrigins role) : reservedParentRole x.constituent = role := by
  simp only [roleOrigins,List.mem_flatMap,List.mem_filter,beq_iff_eq] at present
  obtain ⟨r,⟨hr,hrole⟩,hx⟩ := present
```

```
  rw [reserved_origin_parent r x hx,reservation_parent_role_correct r hr,hrole]

theorem reservation_role_supports_disjoint (a b : Role) (different : a ≠ b)
    (x : Origin) (left : x ∈ roleOrigins a) : x ∉ roleOrigins b := by
  intro right
  exact different ((role_origin_has_unique_parent_role a x left).symm.trans
    (role_origin_has_unique_parent_role b x right))

-- F3. Exact open-control-volume ledger. New outside stock has zero inside
-- increment; its first entry is old-origin transport. Repeated crossings
-- count repeatedly, without manufacturing a fresh origin.
inductive BoundaryEvent where
  | freshInside (carbon : Int)
  | freshOutside (carbon : Int)
  | oldInward (carbon : Int)
  | outward (carbon : Int)
  | internal
  deriving DecidableEq, Repr

structure OpenLedger where
  inside : Int
  fresh : Int
  oldIn : Int
  out : Int
  deriving DecidableEq, Repr

def applyBoundary (s : OpenLedger) : BoundaryEvent → OpenLedger
  | .freshInside n => ⟨s.inside+n,s.fresh+n,s.oldIn,s.out⟩
  | .freshOutside _ => s
  | .oldInward n => ⟨s.inside+n,s.fresh,s.oldIn+n,s.out⟩
  | .outward n => ⟨s.inside-n,s.fresh,s.oldIn,s.out+n⟩
  | .internal => s

def openBalance (s : OpenLedger) : Int := s.inside-s.fresh-s.oldIn+s.out

def applyBoundaryTrace (s : OpenLedger) (events : List BoundaryEvent) : OpenLedger :=
  events.foldl applyBoundary s

theorem one_event_open_balance (s : OpenLedger) (event : BoundaryEvent) :
    openBalance (applyBoundary s event) = openBalance s := by
  cases event <;> simp [openBalance,applyBoundary] <;> omega

theorem arbitrary_trace_open_balance (s : OpenLedger) (events : List BoundaryEvent) :
    openBalance (applyBoundaryTrace s events) = openBalance s := by
  induction events generalizing s with
  | nil => rfl
  | cons event events ih =>
      exact (ih (applyBoundary s event)).trans (one_event_open_balance s event)

theorem exact_open_balance (initial : Int) (events : List BoundaryEvent) :
    let s := applyBoundaryTrace ⟨initial,0,0,0⟩ events
    s.inside = initial+s.fresh+s.oldIn-s.out := by
  have h := arbitrary_trace_open_balance (OpenLedger.mk initial 0 0 0) events
  simp only [openBalance] at h
  dsimp
  omega

theorem outside_admission_no_inside_change (s : OpenLedger) (n : Int) :
    (applyBoundary s (.freshOutside n)).inside = s.inside := rfl

theorem export_return_counts_once_each (s : OpenLedger) (n : Int) :
    let t := applyBoundaryTrace s [.outward n,.oldInward n]
    t.inside = s.inside ∧ t.fresh = s.fresh ∧
    t.oldIn = s.oldIn+n ∧ t.out = s.out+n := by
  simp [applyBoundaryTrace,applyBoundary]

-- F4. Exact natural stoichiometric counts and all-run carbon projections.
structure Counts where
  b : Nat
```

```
  c : Nat
  p : Nat
  w : Nat
  l : Nat
  h : Nat
  r : Nat
  z : Nat
  deriving DecidableEq, Repr

def initialCounts (M : Nat) : Counts := ⟨130*M,30*M,0,0,0,0,0,0⟩
def produce (s : Counts) : Counts :=
  ⟨s.b-10,s.c-2,s.p+1,s.w,s.l+1,s.h+2,s.r+10,s.z+2⟩
def deactivate (s : Counts) : Counts :=
  {s with p := s.p-1, w := s.w+1}
def residueLedger (s : Counts) : Nat := s.b+4*(s.p+s.w+s.l)+s.h
def remnantLedger (s : Counts) : Nat := s.b+s.r
def carrierLedger (s : Counts) : Nat := s.c+s.z

def carbonLedger (s : Counts) (scaffold Cm Cr Cc : Nat) : Nat :=
  scaffold+Cm*residueLedger s+Cr*remnantLedger s+Cc*carrierLedger s

inductive CountStep : Counts → Counts → Prop
  | production (s) (donors : 10 ≤ s.b) (carriers : 2 ≤ s.c) : CountStep s (produce s)
  | deactivation (s) (active : 0 < s.p) : CountStep s (deactivate s)

theorem integer_block_carbon_identity (Cm Cr Cc : Nat) :
    10*(Cm+Cr)+2*Cc = 8*Cm+2*Cm+10*Cr+2*Cc := by
  simp [Nat.mul_add]
  omega

theorem production_preserves_ledgers (s : Counts) (donors : 10 ≤ s.b)
    (carriers : 2 ≤ s.c) :
    residueLedger (produce s) = residueLedger s ∧
    remnantLedger (produce s) = remnantLedger s ∧
    carrierLedger (produce s) = carrierLedger s := by
  simp only [residueLedger,remnantLedger,carrierLedger,produce]
  omega

theorem deactivation_preserves_ledgers (s : Counts) (active : 0 < s.p) :
    residueLedger (deactivate s) = residueLedger s ∧
    remnantLedger (deactivate s) = remnantLedger s ∧
    carrierLedger (deactivate s) = carrierLedger s := by
  simp [residueLedger,remnantLedger,carrierLedger,deactivate]
  omega

theorem count_step_carbon_preserved (s t : Counts) (step : CountStep s t)
    (scaffold Cm Cr Cc : Nat) : carbonLedger t scaffold Cm Cr Cc =
    carbonLedger s scaffold Cm Cr Cc := by
  have h : residueLedger t = residueLedger s ∧ remnantLedger t = remnantLedger s ∧
      carrierLedger t = carrierLedger s := by
    cases step with
    | production donors carriers => exact production_preserves_ledgers s donors carriers
    | deactivation active => exact deactivation_preserves_ledgers s active
  simp [carbonLedger,h.1,h.2.1,h.2.2]

theorem all_count_runs_preserve_carbon {s t : Counts} (run : Execution CountStep s t)
    (scaffold Cm Cr Cc : Nat) : carbonLedger t scaffold Cm Cr Cc =
    carbonLedger s scaffold Cm Cr Cc := by
  induction run with
  | empty => rfl
  | next step _ ih => exact ih.trans (count_step_carbon_preserved _ _ step scaffold Cm Cr Cc)

def countsFromEvents (M B W : Nat) : Counts :=
  ⟨130*M-10*B,30*M-2*B,B-W,W,B,2*B,10*B,2*B⟩

theorem event_count_ledgers (M B W : Nat) (blocks : B ≤ 13*M) (deactivated : W ≤ B) :
    residueLedger (countsFromEvents M B W) = 130*M ∧
    remnantLedger (countsFromEvents M B W) = 130*M ∧
    carrierLedger (countsFromEvents M B W) = 30*M := by
```

```
  simp only [residueLedger,remnantLedger,carrierLedger,countsFromEvents]
  omega

theorem event_count_carrier_floor (M B W : Nat) (blocks : B ≤ 13*M) :
    4*M ≤ (countsFromEvents M B W).c := by
  simp only [countsFromEvents]
  omega

theorem event_count_active_plus_inactive (M B W : Nat) (deactivated : W ≤ B) :
    (countsFromEvents M B W).p + (countsFromEvents M B W).w = B := by
  simp only [countsFromEvents]
  omega

theorem event_count_total_bound (M B W : Nat) (blocks : B ≤ 10*M) (deactivated : W ≤ B) :
    B+W ≤ 20*M := by omega

theorem exhausted_donor_blocks (M B : Nat) (paid : 10*B ≤ 130*M) : B ≤ 13*M := by omega

-- Constructive attainability of every bounded integer event-count state:
-- execute B funded production blocks, then W available deactivations.
theorem event_production_identity (M B W : Nat) (blocks : B+1 ≤ 13*M)
    (deactivated : W ≤ B) :
    produce (countsFromEvents M B W) = countsFromEvents M (B+1) W := by
  simp [produce,countsFromEvents,Counts.mk.injEq]
  omega

theorem event_deactivation_identity (M B W : Nat) :
    deactivate (countsFromEvents M B W) = countsFromEvents M B (W+1) := by
  simp [deactivate,countsFromEvents,Counts.mk.injEq]
  omega

theorem all_production_blocks_exist (M B : Nat) (blocks : B ≤ 13*M) :
    Execution CountStep (initialCounts M) (countsFromEvents M B 0) := by
  induction B with
  | zero =>
      have h : countsFromEvents M 0 0 = initialCounts M := by
        simp [countsFromEvents,initialCounts]
      rw [h]
      exact .empty _
  | succ B ih =>
      have smaller : B ≤ 13*M := by omega
      have donors : 10 ≤ (countsFromEvents M B 0).b := by
        simp only [countsFromEvents]
        omega
      have carriers : 2 ≤ (countsFromEvents M B 0).c := by
        simp only [countsFromEvents]
        omega
      have step := CountStep.production (countsFromEvents M B 0) donors carriers
      rw [event_production_identity M B 0 blocks (by omega)] at step
      exact execution_composition (ih smaller) (.next step (.empty _))

theorem available_deactivations_exist (M B W : Nat) (available : W ≤ B) :
    Execution CountStep (countsFromEvents M B 0) (countsFromEvents M B W) := by
  induction W with
  | zero => exact .empty _
  | succ W ih =>
      have smaller : W ≤ B := by omega
      have active : 0 < (countsFromEvents M B W).p := by
        simp only [countsFromEvents]
        omega
      have step := CountStep.deactivation (countsFromEvents M B W) active
      rw [event_deactivation_identity M B W] at step
      exact execution_composition (ih smaller) (.next step (.empty _))

theorem every_bounded_count_state_reachable (M B W : Nat)
    (blocks : B ≤ 13*M) (deactivated : W ≤ B) :
    Execution CountStep (initialCounts M) (countsFromEvents M B W) := by
  exact execution_composition (all_production_blocks_exist M B blocks)
    (available_deactivations_exist M B W deactivated)
```

```
-- F5. Finite fresh-packet budgets and two-daughter allocation.
def totalCost : List Nat → Nat
  | [] => 0
  | c::cs => c+totalCost cs

theorem concatenated_cost (a b : List Nat) : totalCost (a++b) = totalCost a+totalCost b := by
  induction a with
  | nil => simp [totalCost]
  | cons n a ih => simp [totalCost,ih,Nat.add_assoc]

theorem prefix_withdrawal_budget (a b : List Nat) (budget floor : Nat)
    (paid : totalCost (a++b)+floor ≤ budget) : totalCost a+floor ≤ budget := by
  rw [concatenated_cost] at paid
  omega

theorem bounded_cycle_cost (costs : List Nat) (cap : Nat)
    (bounded : ∀ c ∈ costs, c ≤ cap) : totalCost costs ≤ costs.length*cap := by
  induction costs with
  | nil => simp [totalCost]
  | cons c cs ih =>
      have hc : c ≤ cap := bounded c (by simp)
      have ht : ∀ n ∈ cs, n ≤ cap := fun n hn => bounded n (by simp [hn])
      have h := ih ht
      simp only [totalCost,List.length_cons,Nat.succ_mul]
      omega

theorem finite_cycle_reserve_floor (costs : List Nat) (cap reserve floor : Nat)
    (bounded : ∀ c ∈ costs, c ≤ cap) (funded : costs.length*cap+floor ≤ reserve) :
    floor ≤ reserve-totalCost costs := by
  have h := bounded_cycle_cost costs cap bounded
  omega

def servicedParents : Nat → Nat
  | 0 => 0
  | n+1 => 1+2*servicedParents n

theorem binary_tree_parent_count (n : Nat) : servicedParents n+1 = 2^n := by
  induction n with
  | zero => rfl
  | succ n ih => simp [servicedParents,Nat.pow_succ]; omega

theorem binary_tree_cost (n cap : Nat) : servicedParents n*cap = (2^n-1)*cap := by
  have h := binary_tree_parent_count n
  have eq : servicedParents n = 2^n-1 := by omega
  rw [eq]

theorem daughter_minima_necessary (n n1 n2 discarded m1 m2 : Nat)
    (partition : n1+n2+discarded = n) (left : m1 ≤ n1) (right : m2 ≤ n2) :
    m1+m2 ≤ n := by omega

theorem daughter_minima_sufficient (n m1 m2 : Nat) (enough : m1+m2 ≤ n) :
    ∃ n1 n2 : Nat, n1+n2 = n ∧ m1 ≤ n1 ∧ m2 ≤ n2 := by
  exact ⟨m1,n-m1,by omega,by omega,by omega⟩

theorem one_indivisible_pore_insufficient :
    ¬ ∃ n1 n2 lost : Nat, n1+n2+lost = 1 ∧ 1 ≤ n1 ∧ 1 ≤ n2 := by omega

theorem integer_partition_carbon (n1 n2 lost n C : Nat)
    (partition : n1+n2+lost = n) : n1*C+n2*C+lost*C = n*C := by
  rw [← Nat.add_mul,← Nat.add_mul,partition]

-- F6. The antiparallel reverse-complement identity on arbitrary RNA words.
inductive Base where
  | A | C | G | U
  deriving DecidableEq, Repr

def complement : Base → Base
  | .A => .U
```

```
  | .U => .A
  | .C => .G
  | .G => .C

def reverseComplement (w : List Base) : List Base := (w.map complement).reverse

theorem complement_involutive (b : Base) : complement (complement b) = b := by
  cases b <;> rfl

theorem complement_map_twice (w : List Base) : (w.map complement).map complement = w := by
  induction w with
  | nil => rfl
  | cons b w ih => simp [complement_involutive,ih]

theorem reverse_complement_involutive (w : List Base) :
    reverseComplement (reverseComplement w) = w := by
  simp only [reverseComplement,List.map_reverse,List.reverse_reverse]
  exact complement_map_twice w

theorem reverse_complement_length (w : List Base) : (reverseComplement w).length = w.length := by
  simp [reverseComplement]

-- F7. Rational operating, allocation and transfer algebra.
/-!
## Rational algebra of the operating witness

These are exact pointwise rational lemmas from the displayed reduced model.
They do not assert existence of a real ODE solution, a square-root function,
an integration theorem, physical realization, or a division path.
-/

theorem qDivLe {x y z : Rat} (hy : 0 < y) (h : x ≤ z*y) : x/y ≤ z := by
  apply Rat.le_of_mul_le_mul_right (c := y) ?_ hy
  rw [Rat.div_mul_cancel (by grind : y ≠ 0)]
  exact h

theorem qLeDiv {x y z : Rat} (hy : 0 < y) (h : x*y ≤ z) : x ≤ z/y := by
  apply Rat.le_of_mul_le_mul_right (c := y) ?_ hy
  rw [Rat.div_mul_cancel (by grind : y ≠ 0)]
  exact h

theorem qMulLe {a b c d : Rat}
    (hab : a ≤ b) (hcd : c ≤ d) (ha : 0 ≤ a) (hd : 0 ≤ d) : a*c ≤ b*d :=
  Rat.le_trans (Rat.mul_le_mul_of_nonneg_left hcd ha)
    (Rat.mul_le_mul_of_nonneg_right hab hd)

theorem qUnitProduct {x y : Rat}
    (hx0 : 0 ≤ x) (hx1 : x ≤ 1) (hy0 : 0 ≤ y) (hy1 : y ≤ 1) :
    0 ≤ x*y ∧ x*y ≤ 1 := by
  constructor
  · exact Rat.mul_nonneg hx0 hy0
  · have := qMulLe hx1 hy1 hx0 (by grind : (0:Rat) ≤ 1)
    grind

/-- The model's saturation term. -/
def qSaturation (p : Rat) : Rat := p/(1+p)

theorem q_saturation_bounds (p : Rat) (hp : 0 ≤ p) :
    0 ≤ qSaturation p ∧ qSaturation p ≤ 1 := by
  constructor
  · exact qLeDiv (by grind) (by grind)
  · exact qDivLe (by grind) (by grind)

theorem q_saturation_three (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 3) :
    qSaturation p ≤ 3/4 := by
  apply qDivLe (by grind)
  have hscale : (3/4*(1+p))*4=3+3*p := by grind
  grind

```

```
theorem q_saturation_robust_cap (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 31/10) :
    qSaturation p ≤ 31/41 := by
  apply qDivLe (by grind)
  have hscale : (31/41*(1+p))*41=31+31*p := by grind
  grind

/-- Strict saturation monotonicity, for every rational pair, not a sampled grid. -/
theorem q_saturation_strict (p q : Rat) (hp : 0 ≤ p) (hpq : p < q) :
    qSaturation p < qSaturation q := by
  unfold qSaturation
  have hdp : 0 < 1+p := by grind
  have hdq : 0 < 1+q := by grind
  apply (Rat.div_lt_iff hdp).2
  apply Rat.lt_of_mul_lt_mul_right (c := 1+q) ?_ (by grind)
  have hc : q/(1+q)*(1+q) = q := Rat.div_mul_cancel (by grind)
  have heq : q/(1+q)*(1+p)*(1+q)-p*(1+q) = q-p := by grind
  apply (Rat.lt_iff_sub_pos _ _).2
  rw [heq]
  grind

/-- Availability at nominal parameter values. -/
def qNominalAvailability (b c s : Rat) : Rat :=
  (b/(1+b))*(c/(1/10+c))*(19/20+s/20)

def qNominalFlux (p b c s : Rat) : Rat :=
  (10+20*qSaturation p)*qNominalAvailability b c s

def qMu : Rat := 2850/3131

def qDelta : Rat := 3147/6262

def qNominalSpeed (p : Rat) : Rat := qMu*(1+2*p/(1+p))-p

theorem q_nominal_mu_exact :
    (19/20:Rat)*(30/31)*(100/101) = qMu := by
  unfold qMu
  grind

theorem q_nominal_mu_positive : 0 < qMu := by
  unfold qMu
  grind

theorem q_nominal_availability_unit (b c s : Rat)
    (hb : 0 ≤ b) (hc : 0 ≤ c) (hs0 : 0 ≤ s) (hs1 : s ≤ 1) :
    0 ≤ qNominalAvailability b c s ∧ qNominalAvailability b c s ≤ 1 := by
  have hb0 : 0 ≤ b/(1+b) := qLeDiv (by grind) (by grind)
  have hb1 : b/(1+b) ≤ 1 := qDivLe (by grind) (by grind)
  have hc0 : 0 ≤ c/(1/10+c) := qLeDiv (by grind) (by grind)
  have hc1 : c/(1/10+c) ≤ 1 := qDivLe (by grind) (by grind)
  have hs20 : s/20*20=s := Rat.div_mul_cancel (by decide +kernel)
  have hbc := qUnitProduct hb0 hb1 hc0 hc1
  exact qUnitProduct hbc.1 hbc.2 (by grind) (by grind)

theorem q_nominal_availability_lower (b c s : Rat)
    (hb : 30 ≤ b) (hc : 10 ≤ c) (hs : 0 ≤ s) :
    qMu ≤ qNominalAvailability b c s := by
  have hb' : 30/31 ≤ b/(1+b) := by
    apply qLeDiv (by grind)
    have hscale : (30/31*(1+b))*31=30+30*b := by grind
    grind
  have hc' : 100/101 ≤ c/(1/10+c) := by
    apply qLeDiv (by grind)
    have hscale : (100/101*(1/10+c))*101=10+100*c := by grind
    grind
  have hs20 : s/20*20=s := Rat.div_mul_cancel (by decide +kernel)
  have hbc := qMulLe hb' hc' (by grind : (0:Rat) ≤ 30/31) (by grind)
  have hall := qMulLe hbc (by grind : (19/20:Rat) ≤ 19/20+s/20)
    (by decide +kernel : (0:Rat) ≤ (30/31)*(100/101)) (by grind)
  have heq : (30/31:Rat)*(100/101)*(19/20)=qMu := by unfold qMu; decide +kernel
```

```
  change qMu ≤ _
  rw [← heq]
  exact hall

/-- Pointwise crude and sharpened nominal production bounds. -/
theorem q_nominal_flux_bounds (p b c s : Rat)
    (hp : 0 ≤ p) (hu : p ≤ 3) (hb : 0 ≤ b) (hc : 0 ≤ c)
    (hs0 : 0 ≤ s) (hs1 : s ≤ 1) :
    0 ≤ qNominalFlux p b c s ∧ qNominalFlux p b c s ≤ 25 := by
  have ha := q_nominal_availability_unit b c s hb hc hs0 hs1
  have hp0 := (q_saturation_bounds p hp).1
  have hp1 := q_saturation_three p hp hu
  have hf0 : 0 ≤ 10+20*qSaturation p := by grind
  have hf1 : 10+20*qSaturation p ≤ 25 := by grind
  constructor
  · exact Rat.mul_nonneg hf0 ha.1
  · have := qMulLe hf1 ha.2 hf0 (by grind : (0:Rat) ≤ 1)
    unfold qNominalFlux
    grind

theorem q_nominal_flux_lower (p b c s : Rat)
    (hp : 0 ≤ p) (hb : 30 ≤ b) (hc : 10 ≤ c) (hs : 0 ≤ s) :
    10*qMu ≤ qNominalFlux p b c s := by
  have ha := q_nominal_availability_lower b c s hb hc hs
  have hm := q_nominal_mu_positive
  have hp0 := (q_saturation_bounds p hp).1
  have hf : 10 ≤ 10+20*qSaturation p := by grind
  exact qMulLe hf ha (by grind) (by grind)

theorem q_nominal_flux_strict_feedback (p q b c s : Rat)
    (hp : 0 ≤ p) (hpq : p < q) (hb : 0 < b) (hc : 0 < c) (hs : 0 ≤ s) :
    qNominalFlux p b c s < qNominalFlux q b c s := by
  have hsat := q_saturation_strict p q hp hpq
  have hb' : 0 < b/(1+b) := (Rat.lt_div_iff (by grind)).2 (by grind)
  have hc' : 0 < c/(1/10+c) := (Rat.lt_div_iff (by grind)).2 (by grind)
  have hs20 : s/20*20=s := Rat.div_mul_cancel (by decide +kernel)
  have ha : 0 < qNominalAvailability b c s :=
    Rat.mul_pos (Rat.mul_pos hb' hc') (by grind)
  exact Rat.mul_lt_mul_of_pos_right (by grind) ha

/-- Exact nonnegative factorization replaces a calculus concavity lemma here. -/
theorem q_nominal_speed_lower (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 3/2) :
    qDelta ≤ qNominalSpeed p := by
  unfold qDelta qNominalSpeed qMu
  have hd : 0 < 1+p := by grind
  have hcancel : (2*p)/(1+p)*(1+p) = 2*p := Rat.div_mul_cancel (by grind)
  have hprod : 0 ≤ (3/2-p)*(p+851/3131) := Rat.mul_nonneg (by grind) (by grind)
  have halg : (((2850/3131:Rat)*(1+2*p/(1+p))-p)-3147/6262)*(1+p) =
      (3/2-p)*(p+851/3131) := by grind
  have hm : 0*(1+p) ≤ (((2850/3131:Rat)*(1+2*p/(1+p))-p)-3147/6262)*(1+p) := by grind
  have hb := Rat.le_of_mul_le_mul_right hm hd
  grind

theorem q_nominal_speed_positive (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 3/2) :
    0 < qNominalSpeed p := by
  have := q_nominal_speed_lower p hp hu
  unfold qDelta at *
  grind

theorem q_nominal_crossing_constants :
    0 < qDelta ∧ (3/2:Rat)/qDelta = 3131/1049 ∧
    (3131/1049:Rat) < 597/200 ∧ (597/200:Rat) < 4 := by
  unfold qDelta
  grind

theorem q_nominal_margin_speed (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 8/5) :
    (87626/203515:Rat) ≤ qNominalSpeed p := by
  unfold qNominalSpeed qMu
  have hd : 0 < 1+p := by grind
```

```
  have hcancel : (2*p)/(1+p)*(1+p) = 2*p := Rat.div_mul_cancel (by grind)
  have hprod : 0 ≤ (8/5-p)*(p+12203/40703) := Rat.mul_nonneg (by grind) (by grind)
  have halg : (((2850/3131:Rat)*(1+2*p/(1+p))-p)-87626/203515)*(1+p) =
      (8/5-p)*(p+12203/40703) := by grind
  have hm : 0*(1+p) ≤ (((2850/3131:Rat)*(1+2*p/(1+p))-p)-87626/203515)*(1+p) := by grind
  have hb := Rat.le_of_mul_le_mul_right hm hd
  grind

theorem q_nominal_margin_constants :
    (0:Rat) < 87626/203515 ∧
    (8/5:Rat)/(87626/203515) = 162812/43813 ∧
    (162812/43813:Rat) < 3717/1000 ∧
    (8/5:Rat)-(3/2) = 1/10 := by grind

/-- Exact extensive ledgers for arbitrary rational cumulative admission. -/
theorem q_nominal_material_ledgers (S p w l h r z : Rat)
    (hpw : p+w = S/10) (hl : l = S/10) (hh : h = S/5)
    (hr : r = S) (hz : z = S/5) :
    (130-S)+4*(p+w+l)+h = 130 ∧
    (130-S)+r = 130 ∧ (30-S/5)+z = 30 ∧
    500+4*(p+w)+r+z = 500+8*S/5 ∧
    20+l+h = 20+3*S/10 := by grind

theorem q_nominal_operating_ledger_bounds (S : Rat) (h0 : 0 ≤ S) (h1 : S ≤ 100) :
    30 ≤ 130-S ∧ 10 ≤ 30-S/5 ∧
    500 ≤ 500+8*S/5 ∧ 500+8*S/5 ≤ 660 ∧
    20 ≤ 20+3*S/10 ∧ 20+3*S/10 ≤ 50 := by
  have h5 : S/5*5=S := Rat.div_mul_cancel (by decide +kernel)
  have h8 : (8*S/5)*5=8*S := Rat.div_mul_cancel (by decide +kernel)
  have h3 : (3*S/10)*10=3*S := Rat.div_mul_cancel (by decide +kernel)
  grind

theorem q_nominal_global_ledger_bounds (S : Rat) (_h0 : 0 ≤ S) (h1 : S ≤ 130) :
    0 ≤ 130-S ∧ 4 ≤ 30-S/5 ∧
    500+8*S/5 ≤ 708 ∧ 20+3*S/10 ≤ 59 := by
  have h5 : S/5*5=S := Rat.div_mul_cancel (by decide +kernel)
  have h8 : (8*S/5)*5=8*S := Rat.div_mul_cancel (by decide +kernel)
  have h3 : (3*S/10)*10=3*S := Rat.div_mul_cancel (by decide +kernel)
  grind

theorem q_nominal_energy_floor (t P : Rat) (ht : t ≤ 4) (hP : P ≤ 30*t) :
    40 ≤ 160-P := by grind

/-- Rational upper-screen and boundary-sign arithmetic; pi and roots are not defined. -/
theorem q_nominal_water_geometry_constants :
    (20/100:Rat)-1/10 = 1/10 ∧
    (50/600:Rat)+1/(600*600)-1/10 = -5999/360000 ∧
    (-5999/360000:Rat) < 0 ∧
    (59/600:Rat)+1/(600*600)-1/10 = -599/360000 ∧
    (-599/360000:Rat) < 0 ∧
    (36*(22/7)*600*600/(500*500*500):Rat) < 1/3 ∧
    (300/660:Rat) = 5/11 ∧ (5/11:Rat) > 2/5 := by decide +kernel

/-- Algebraic link between the actual nominal vector field and its uniform speed. -/
theorem q_nominal_vectorfield_lower (p b c s : Rat)
    (hp : 0 ≤ p) (hb : 30 ≤ b) (hc : 10 ≤ c) (hs : 0 ≤ s) :
    qNominalSpeed p ≤ qNominalFlux p b c s/10-p := by
  have ha := q_nominal_availability_lower b c s hb hc hs
  have hp0 := (q_saturation_bounds p hp).1
  have hm := Rat.mul_le_mul_of_nonneg_left ha (by grind : 0 ≤ 10+20*qSaturation p)
  have hid : (10+20*qSaturation p)*qMu = 10*(qNominalSpeed p+p) := by
    unfold qSaturation qNominalSpeed
    grind
  have hdiv : qNominalFlux p b c s/10*10=qNominalFlux p b c s :=
    Rat.div_mul_cancel (by decide +kernel)
  change (10+20*qSaturation p)*qMu ≤ qNominalFlux p b c s at hm
  grind

theorem q_nominal_vectorfield_speed (p b c s : Rat)
```

```
    (hp : 0 ≤ p) (hu : p ≤ 3/2) (hb : 30 ≤ b) (hc : 10 ≤ c) (hs : 0 ≤ s) :
    qDelta ≤ qNominalFlux p b c s/10-p :=
  Rat.le_trans (q_nominal_speed_lower p hp hu)
    (q_nominal_vectorfield_lower p b c s hp hb hc hs)

/-! ## Uniform robust-box rational certificate -/

def qRobustMu : Rat := 717500/799663

def qRobustDelta : Rat := 70245111/159932600

def qRobustSpeed (p : Rat) : Rat :=
  qRobustMu*(99/100+(99/50)*p/(1+p))-(101/100)*p

def qRobustAvailability (b c s Kb Kc eta : Rat) : Rat :=
  (b/(Kb+b))*(c/(Kc+c))*(19/20+s/20+eta)

def qRobustFlux (p b c s a0 a1 Kb Kc eta : Rat) : Rat :=
  (a0+a1*qSaturation p)*qRobustAvailability b c s Kb Kc eta

theorem q_robust_switch_multiplier (s eta : Rat)
    (hs0 : 0 ≤ s) (hs1 : s ≤ 1) (he0 : -1/200 ≤ eta) (he1 : eta ≤ 1/200) :
    189/200 ≤ 19/20+s/20+eta ∧ 19/20+s/20+eta ≤ 201/200 := by
  have hs20 : s/20*20=s := Rat.div_mul_cancel (by decide +kernel)
  grind

theorem q_robust_mu_exact :
    (189/200:Rat)*(25/(25+101/100))*((41/5)/(41/5+101/1000)) = qRobustMu := by
  unfold qRobustMu
  decide +kernel

theorem q_robust_availability_lower (b c s Kb Kc eta : Rat)
    (hb : 25 ≤ b) (hc : 41/5 ≤ c) (hs : 0 ≤ s)
    (hKb0 : 0 < Kb) (hKb1 : Kb ≤ 101/100)
    (hKc0 : 0 < Kc) (hKc1 : Kc ≤ 101/1000) (heta : -1/200 ≤ eta) :
    qRobustMu ≤ qRobustAvailability b c s Kb Kc eta := by
  have hb' : 2500/2601 ≤ b/(Kb+b) := by
    apply qLeDiv (by grind)
    have hscale : (2500/2601*(Kb+b))*2601=2500*Kb+2500*b := by grind
    grind
  have hc' : 8200/8301 ≤ c/(Kc+c) := by
    apply qLeDiv (by grind)
    have hscale : (8200/8301*(Kc+c))*8301=8200*Kc+8200*c := by grind
    grind
  have hs20 : s/20*20=s := Rat.div_mul_cancel (by decide +kernel)
  have hbc := qMulLe hb' hc' (by decide +kernel : (0:Rat) ≤ 2500/2601) (by grind)
  have hall := qMulLe hbc (by grind : (189/200:Rat) ≤ 19/20+s/20+eta)
    (by decide +kernel : (0:Rat) ≤ (2500/2601)*(8200/8301)) (by grind)
  have heq : (2500/2601:Rat)*(8200/8301)*(189/200)=qRobustMu := by
    unfold qRobustMu
    decide +kernel
  change qRobustMu ≤ _
  rw [← heq]
  exact hall

/-- Uniform speed bound throughout [0,3/2], established by a factorization. -/
theorem q_robust_speed_lower (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 3/2) :
    qRobustDelta ≤ qRobustSpeed p := by
  unfold qRobustSpeed qRobustDelta qRobustMu
  have hd : 0 < 1+p := by grind
  have hcancel : ((99/50)*p)/(1+p)*(1+p) = (99/50)*p := Rat.div_mul_cancel (by grind)
  have hcscale : ((101/100)*p)*100=101*p := by grind
  have hright : 0 ≤ (101/100)*p+23939963/79966300 := by grind
  have hprod : 0 ≤ (3/2-p)*((101/100)*p+23939963/79966300) :=
    Rat.mul_nonneg (by grind) hright
  have halg : (((717500/799663:Rat)*(99/100+(99/50)*p/(1+p))-(101/100)*p)-
      70245111/159932600)*(1+p) = (3/2-p)*((101/100)*p+23939963/79966300) := by grind
  have hm : 0*(1+p) ≤ (((717500/799663:Rat)*(99/100+(99/50)*p/(1+p))-(101/100)*p)-
      70245111/159932600)*(1+p) := by grind
```

```
  have hb := Rat.le_of_mul_le_mul_right hm hd
  grind

theorem q_robust_speed_positive (p : Rat) (hp : 0 ≤ p) (hu : p ≤ 3/2) :
    0 < qRobustSpeed p := by
  have := q_robust_speed_lower p hp hu
  unfold qRobustDelta at *
  grind

theorem q_robust_crossing_constants :
    0 < qRobustDelta ∧ (3/2:Rat)/qRobustDelta = 239898900/70245111 ∧
    (239898900/70245111:Rat) < 427/125 ∧ (427/125:Rat) < 4 := by
  unfold qRobustDelta
  decide +kernel

/-- Crude cap, inward active-bound sign, and sharpened robust flux cap. -/
theorem q_robust_flux_cap_constants :
    (101/10+101/5:Rat)*(201/200) = 60903/2000 ∧
    (60903/2000:Rat)/10-(99/100)*(31/10) < 0 ∧
    ((101/10:Rat)+(101/5)*(31/41))*(201/200) = 2091003/82000 ∧
    (2091003/82000:Rat) < 26 := by decide +kernel

theorem q_robust_material_bounds (S b0 c0 : Rat)
    (_hS0 : 0 ≤ S) (hS1 : S ≤ 104) (hb : 129 ≤ b0) (hc : 29 ≤ c0) :
    25 ≤ b0-S ∧ 41/5 ≤ c0-S/5 ∧
    501+(41/100+101/100+101/500)*S ≤ 83711/125 ∧
    201/10+3*S/10 ≤ 513/10 := by
  have hS5 : S/5*5=S := Rat.div_mul_cancel (by decide +kernel)
  have hS10 : (3*S/10)*10=3*S := Rat.div_mul_cancel (by decide +kernel)
  have ha : ((41/100+101/100+101/500)*S)*500=811*S := by grind
  grind

theorem q_robust_area_water_geometry_constants :
    (501:Rat)+(41/100+101/100+101/500)*104 = 83711/125 ∧
    (83711/125:Rat) < 670 ∧
    (199/10:Rat)/100-101/1000-1/1000 = 97/1000 ∧ (0:Rat) < 97/1000 ∧
    (513/10:Rat)/600+(21/10)*(21/10)/(4*(99/100)*600*600)-99/1000+1/1000 < 0 ∧
    (36*(22/7)*600*600/(499*499*499):Rat) < 1/3 ∧
    (300/670:Rat) > 2/5 := by decide +kernel

theorem q_robust_ablation_energy_constants :
    (101/10:Rat)*(201/200) = 20301/2000 ∧
    ((20301/2000:Rat)/10)/(99/100) < 513/500 ∧
    (513/500:Rat) < 3/2 ∧
    (160:Rat)-4*(26+2+31/10) = 178/5 ∧ (178/5:Rat) > 35 := by decide +kernel

theorem q_robust_energy_floor (t used : Rat)
    (ht : t ≤ 4) (hu : used ≤ (311/10)*t) : 178/5 ≤ 160-used := by
  have hscale : ((311/10)*t)*10=311*t := by grind
  grind

/-- Error transfer at the strict nominal terminal margin. -/
theorem q_terminal_margin_transfer (nominal actual epsilon : Rat)
    (hn : 8/5 ≤ nominal) (he : epsilon < 1/10) (ha : nominal-actual ≤ epsilon) :
    3/2 < actual := by grind

/-! ## Rounding-error, reset, and return-map algebra -/

/-- Conditional fractional-part error bound. No floor or scheduling is postulated. -/
theorem q_fractional_count_error (epsB epsW M : Rat)
    (hB0 : 0 ≤ epsB) (hB1 : epsB < 1)
    (hW0 : 0 ≤ epsW) (hW1 : epsW < 1) (hM : 0 < M) :
    -1/M < (epsW-epsB)/M ∧ (epsW-epsB)/M < 1/M := by
  have hc1 : (-1/M)*M=(-1:Rat) := Rat.div_mul_cancel (by grind)
  have hc2 : ((epsW-epsB)/M)*M=epsW-epsB := Rat.div_mul_cancel (by grind)
  have hc3 : (1/M)*M=(1:Rat) := Rat.div_mul_cancel (by grind)
  constructor
  · apply Rat.lt_of_mul_lt_mul_right (c := M) ?_ (by grind)
    rw [hc1, hc2]
```

```
    grind
  · apply Rat.lt_of_mul_lt_mul_right (c := M) ?_ (by grind)
    rw [hc2, hc3]
    grind

theorem q_inverse_scale_margin (M : Rat) (hM : 10 < M) : (1:Rat)/M < 1/10 := by
  apply (Rat.div_lt_iff (by grind : 0 < M)).2
  have hscale : ((1/10)*M)*10=M := by grind
  grind

/-- Strict molecular-count threshold transfer under the stated fractional errors. -/
theorem q_count_threshold_transfer (p counted epsB epsW M : Rat)
    (hp : 8/5 ≤ p) (hM : 10 < M)
    (hB0 : 0 ≤ epsB) (hB1 : epsB < 1)
    (hW0 : 0 ≤ epsW) (hW1 : epsW < 1)
    (hcount : counted = p+(epsW-epsB)/M) : 3/2 < counted := by
  have herr := q_fractional_count_error epsB epsW M hB0 hB1 hW0 hW1 (by grind)
  have hsmall := q_inverse_scale_margin M hM
  have hneg : (-1:Rat)/M = -(1/M) := by grind
  grind

/-- A strictly consumed reserve cannot return to its previous value. -/
theorem q_no_productive_reserve_return (before after cost : Rat)
    (hcost : 0 < cost) (hwithdraw : after=before-cost) : after < before ∧ after ≠ before := by
  grind

theorem q_no_positive_counter_return (before after increment : Rat)
    (hinc : 0 < increment) (hcounter : after=before+increment) :
    before < after ∧ after ≠ before := by grind

/-- Normalization, not the unnormalized powers, conserves parent area. -/
theorem q_normalized_daughter_areas (x y A : Rat) (hx : 0 < x) (hy : 0 < y) (hA : 0 ≤ A) :
    0 ≤ (x/(x+y))*A ∧ 0 ≤ (y/(x+y))*A ∧
    x/(x+y)+y/(x+y)=1 ∧ (x/(x+y))*A+(y/(x+y))*A=A := by
  have hden : x+y ≠ 0 := by grind
  have hx' : 0 ≤ x/(x+y) := qLeDiv (by grind) (by grind)
  have hy' : 0 ≤ y/(x+y) := qLeDiv (by grind) (by grind)
  have hi : (x+y)*(x+y)^-1=1 := Rat.mul_inv_cancel _ hden
  refine ⟨Rat.mul_nonneg hx' hA, Rat.mul_nonneg hy' hA, ?_, ?_⟩ <;> grind

theorem q_proportional_allocation_conservation (share amount : Rat)
    (hs0 : 0 ≤ share) (hs1 : share ≤ 1) (ha : 0 ≤ amount) :
    0 ≤ share*amount ∧ 0 ≤ (1-share)*amount ∧
    share*amount+(1-share)*amount=amount := by
  refine ⟨Rat.mul_nonneg hs0 ha, Rat.mul_nonneg (by grind) ha, ?_⟩
  grind

/-- Proportional lumen amounts and volume have the same concentration. -/
theorem q_lumen_concentration_preserved (alpha n V : Rat)
    (ha : 0 < alpha) (hV : 0 < V) : (alpha*n)/(alpha*V)=n/V := by
  have hnV : V ≠ 0 := by grind
  have hnA : alpha ≠ 0 := by grind
  have hnaV : alpha*V ≠ 0 := Rat.ne_of_gt (Rat.mul_pos ha hV)
  have hc1 : ((alpha*n)/(alpha*V))*(alpha*V)=alpha*n := Rat.div_mul_cancel hnaV
  have hc2 : (n/V)*V=n := Rat.div_mul_cancel hnV
  have hiA : alpha*alpha^-1=1 := Rat.mul_inv_cancel _ hnA
  have hiV : V*V^-1=1 := Rat.mul_inv_cancel _ hnV
  grind

/-- A distinct membrane-charge share preserves charge density only if it equals
    the lumen-volume share (for nonzero parent charge and positive volumes). -/
theorem q_charge_density_share_iff (alpha beta Q V : Rat)
    (ha : 0 < alpha) (hQ : 0 < Q) (hV : 0 < V) :
    (beta*Q)/(alpha*V)=Q/V ↔ beta=alpha := by
  have hnV : V ≠ 0 := by grind
  have hnQ : Q ≠ 0 := by grind
  have hnaV : alpha*V ≠ 0 := Rat.ne_of_gt (Rat.mul_pos ha hV)
  have hc1 : ((beta*Q)/(alpha*V))*(alpha*V)=beta*Q := Rat.div_mul_cancel hnaV
  have hc2 : (Q/V)*V=Q := Rat.div_mul_cancel hnV
```

```
  have hiV : V*V⁻¹=1 := Rat.mul_inv_cancel _ hnV
  have hiQ : Q*Q⁻¹=1 := Rat.mul_inv_cancel _ hnQ
  constructor
  · intro heq
    grind
  · intro heq
    subst beta
    exact q_lumen_concentration_preserved alpha Q V ha hV

theorem q_asymmetric_charge_density_example :
    (((1/2:Rat)*2)/((1/3)*200)) = 3/200 ∧
    ((2:Rat)/200) = 1/100 ∧ (3/200:Rat) ≠ 1/100 := by decide +kernel

/-- Exact scalar linear-attraction interval; the omitted lower slope bound matters. -/
theorem q_return_slope_interval (g : Rat) :
    (-1 < (1+g)/2 ∧ (1+g)/2 < 1) ↔ (-3 < g ∧ g < 1) := by
  have hscale : ((1+g)/2)*2=1+g := Rat.div_mul_cancel (by decide +kernel)
  grind

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

/-- Quantitative strict self-map test after all geometric/norm estimates are supplied. -/
theorem q_robust_selfmap_margin (radius centerError q delta eps distance : Rat)
    (hmargin : centerError+q*radius ≤ radius-delta)
    (heps : eps < delta) (hdist : distance ≤ eps+centerError+q*radius) :
    distance < radius := by grind

theorem q_robust_contraction_margin (q epsD : Rat) (he : epsD < 1-q) : q+epsD < 1 := by
  grind

/-- Rationalized Donnan estimate conditional on the defining root identity.
    This proves the algebraic inequality, not existence of a real square root. -/
theorem q_donnan_rationalized_bound (x salt root : Rat)
    (hs : 0 < salt) (hr : 2*salt ≤ root)
    (hroot : root*root=x*x+4*salt*salt) :
    0 ≤ root-2*salt ∧ root-2*salt ≤ (x*x)/(4*salt) := by
  have hnonneg : 0 ≤ root-2*salt := by grind
  have hm := Rat.mul_le_mul_of_nonneg_left
    (by grind : 4*salt ≤ root+2*salt) hnonneg
  have halg : (root-2*salt)*(root+2*salt)=x*x := by grind
  constructor
  · exact hnonneg
  · exact qLeDiv (by grind) (by grind)

/-- General positive-reserve saturation factor lies in the unit interval. -/
theorem q_reserve_ratio_unit (reserve halfSaturation : Rat)
    (hr : 0 ≤ reserve) (hK : 0 < halfSaturation) :
    0 ≤ reserve/(halfSaturation+reserve) ∧ reserve/(halfSaturation+reserve) ≤ 1 := by
  constructor
  · exact qLeDiv (by grind) (by grind)
  · exact qDivLe (by grind) (by grind)

theorem q_robust_availability_upper (b c s Kb Kc eta : Rat)
    (hb : 0 ≤ b) (hc : 0 ≤ c) (hs0 : 0 ≤ s) (hs1 : s ≤ 1)
    (hKb : 0 < Kb) (hKc : 0 < Kc) (he0 : -1/200 ≤ eta) (he1 : eta ≤ 1/200) :
    0 ≤ qRobustAvailability b c s Kb Kc eta ∧
    qRobustAvailability b c s Kb Kc eta ≤ 201/200 := by
  have hbr := q_reserve_ratio_unit b Kb hb hKb
  have hcr := q_reserve_ratio_unit c Kc hc hKc
  have hbc := qUnitProduct hbr.1 hbr.2 hcr.1 hcr.2
  have hs := q_robust_switch_multiplier s eta hs0 hs1 he0 he1
  constructor
  · exact Rat.mul_nonneg hbc.1 (by grind)
  · have hh := qMulLe hbc.2 hs.2 hbc.1 (by decide +kernel : (0:Rat) ≤ 201/200)
    change _ ≤ _
    have hmul : (1:Rat)*(201/200)=201/200 := by decide +kernel
    rw [hmul] at hh
    exact hh
```

```
/-- The robust J≤26 claim as a universal inequality for the actual rational flux. -/
theorem q_robust_flux_upper (p b c s a0 a1 Kb Kc eta : Rat)
    (hp0 : 0 ≤ p) (hp1 : p ≤ 31/10) (hb : 0 ≤ b) (hc : 0 ≤ c)
    (hs0 : 0 ≤ s) (hs1 : s ≤ 1) (ha00 : 0 ≤ a0) (ha01 : a0 ≤ 101/10)
    (ha10 : 0 ≤ a1) (ha11 : a1 ≤ 101/5)
    (hKb : 0 < Kb) (hKc : 0 < Kc) (he0 : -1/200 ≤ eta) (he1 : eta ≤ 1/200) :
    0 ≤ qRobustFlux p b c s a0 a1 Kb Kc eta ∧
    qRobustFlux p b c s a0 a1 Kb Kc eta < 26 := by
  have ha := q_robust_availability_upper b c s Kb Kc eta hb hc hs0 hs1 hKb hKc he0 he1
  have hsat0 := (q_saturation_bounds p hp0).1
  have hsat1 := q_saturation_robust_cap p hp0 hp1
  have hmul0 := Rat.mul_nonneg ha10 hsat0
  have hmul1 := qMulLe ha11 hsat1 ha10 (by decide +kernel : (0:Rat) ≤ 31/41)
  have hf0 : 0 ≤ a0+a1*qSaturation p := by grind
  have hf1 : a0+a1*qSaturation p ≤ (101/10)+(101/5)*(31/41) := by grind
  have hbound := qMulLe hf1 ha.2 hf0 (by decide +kernel : (0:Rat) ≤ 201/200)
  have hconstant : ((101/10:Rat)+(101/5)*(31/41))*(201/200) = 2091003/82000 := by decide +kernel
  have hstrict : (2091003/82000:Rat) < 26 := by decide +kernel
  constructor
  · exact Rat.mul_nonneg hf0 ha.1
  · change qRobustFlux p b c s a0 a1 Kb Kc eta ≤ _ at hbound
    rw [hconstant] at hbound
    grind

/-- The robust active vector field lies above its uniform comparison speed. -/
theorem q_robust_vectorfield_lower (p b c s a0 a1 k Kb Kc eta : Rat)
    (hp : 0 ≤ p) (hb : 25 ≤ b) (hc : 41/5 ≤ c) (hs : 0 ≤ s)
    (ha0 : 99/10 ≤ a0) (ha1 : 99/5 ≤ a1) (hk : k ≤ 101/100)
    (hKb0 : 0 < Kb) (hKb1 : Kb ≤ 101/100)
    (hKc0 : 0 < Kc) (hKc1 : Kc ≤ 101/1000) (heta : -1/200 ≤ eta) :
    qRobustSpeed p ≤ qRobustFlux p b c s a0 a1 Kb Kc eta/10-k*p := by
  have ha := q_robust_availability_lower b c s Kb Kc eta hb hc hs hKb0 hKb1 hKc0 hKc1 heta
  have hsat0 := (q_saturation_bounds p hp).1
  have hmul := Rat.mul_le_mul_of_nonneg_right ha1 hsat0
  have hmul0 : 0 ≤ (99/5)*qSaturation p :=
    Rat.mul_nonneg (by decide +kernel) hsat0
  have hf : 99/10+(99/5)*qSaturation p ≤ a0+a1*qSaturation p := by grind
  have hmupos : 0 < qRobustMu := by unfold qRobustMu; decide +kernel
  have hf0 : 0 ≤ 99/10+(99/5)*qSaturation p := by grind
  have hprod := qMulLe hf ha hf0 (by grind)
  have hkprod := Rat.mul_le_mul_of_nonneg_right hk hp
  have hid : (99/10+(99/5)*qSaturation p)*qRobustMu =
      10*(qRobustSpeed p+(101/100)*p) := by
    unfold qSaturation qRobustSpeed
    grind
  have hdiv : (qRobustFlux p b c s a0 a1 Kb Kc eta/10)*10=
      qRobustFlux p b c s a0 a1 Kb Kc eta := Rat.div_mul_cancel (by decide +kernel)
  change (99/10+(99/5)*qSaturation p)*qRobustMu ≤ qRobustFlux p b c s a0 a1 Kb Kc eta at hprod
  grind

theorem q_robust_vectorfield_speed (p b c s a0 a1 k Kb Kc eta : Rat)
    (hp : 0 ≤ p) (hu : p ≤ 3/2) (hb : 25 ≤ b) (hc : 41/5 ≤ c) (hs : 0 ≤ s)
    (ha0 : 99/10 ≤ a0) (ha1 : 99/5 ≤ a1) (hk : k ≤ 101/100)
    (hKb0 : 0 < Kb) (hKb1 : Kb ≤ 101/100)
    (hKc0 : 0 < Kc) (hKc1 : Kc ≤ 101/1000) (heta : -1/200 ≤ eta) :
    qRobustDelta ≤ qRobustFlux p b c s a0 a1 Kb Kc eta/10-k*p :=
  Rat.le_trans (q_robust_speed_lower p hp hu)
    (q_robust_vectorfield_lower p b c s a0 a1 k Kb Kc eta hp hb hc hs ha0 ha1 hk hKb0 hKb1 hKc0 hKc1 heta)

end Article77

-- Complete per-theorem transitive kernel-dependency audit.
#print axioms Article77.execution_composition
#print axioms Article77.execution_invariant
#print axioms Article77.copy_prefix_injective
#print axioms Article77.different_admissions_disjoint
#print axioms Article77.owner_partition_exhaustive
#print axioms Article77.owner_partition_disjoint
```

```
#print axioms Article77.moved_support_owns
#print axioms Article77.moved_support_frame
#print axioms Article77.different_owner_updates_commute
#print axioms Article77.initial_material_safe
#print axioms Article77.material_step_preserves
#print axioms Article77.all_material_runs_preserve_ancestry
#print axioms Article77.released_product_origins_unique
#print axioms Article77.material_admitted_monotone
#print axioms Article77.old_origins_never_deleted
#print axioms Article77.initial_typed_material_safe
#print axioms Article77.typed_material_step_preserves
#print axioms Article77.every_release_is_nonempty_and_traced
#print axioms Article77.allocation_cannot_release
#print axioms Article77.released_cannot_release_again
#print axioms Article77.disjoint_writes_can_disable_guard
#print axioms Article77.five_neutral_precursor_formulas
#print axioms Article77.substitution_hydrogen_balance
#print axioms Article77.reservation_has_38_constituents
#print axioms Article77.each_reserved_support_unique
#print axioms Article77.reserved_origin_parent
#print axioms Article77.reservation_parent_names_distinct
#print axioms Article77.reservation_has_810_distinct_carbons
#print axioms Article77.reservation_role_counts
#print axioms Article77.reservation_parent_role_correct
#print axioms Article77.role_origin_has_unique_parent_role
#print axioms Article77.reservation_role_supports_disjoint
#print axioms Article77.one_event_open_balance
#print axioms Article77.arbitrary_trace_open_balance
#print axioms Article77.exact_open_balance
#print axioms Article77.outside_admission_no_inside_change
#print axioms Article77.export_return_counts_once_each
#print axioms Article77.integer_block_carbon_identity
#print axioms Article77.production_preserves_ledgers
#print axioms Article77.deactivation_preserves_ledgers
#print axioms Article77.count_step_carbon_preserved
#print axioms Article77.all_count_runs_preserve_carbon
#print axioms Article77.event_count_ledgers
#print axioms Article77.event_count_carrier_floor
#print axioms Article77.event_count_active_plus_inactive
#print axioms Article77.event_count_total_bound
#print axioms Article77.exhausted_donor_blocks
#print axioms Article77.event_production_identity
#print axioms Article77.event_deactivation_identity
#print axioms Article77.all_production_blocks_exist
#print axioms Article77.available_deactivations_exist
#print axioms Article77.every_bounded_count_state_reachable
#print axioms Article77.concatenated_cost
#print axioms Article77.prefix_withdrawal_budget
#print axioms Article77.bounded_cycle_cost
#print axioms Article77.finite_cycle_reserve_floor
#print axioms Article77.binary_tree_parent_count
#print axioms Article77.binary_tree_cost
#print axioms Article77.daughter_minima_necessary
#print axioms Article77.daughter_minima_sufficient
#print axioms Article77.one_indivisible_pore_insufficient
#print axioms Article77.integer_partition_carbon
#print axioms Article77.complement_involutive
#print axioms Article77.complement_map_twice
#print axioms Article77.reverse_complement_involutive
#print axioms Article77.reverse_complement_length
#print axioms Article77.qDivLe
#print axioms Article77.qLeDiv
#print axioms Article77.qMulLe
#print axioms Article77.qUnitProduct
#print axioms Article77.q_saturation_bounds
#print axioms Article77.q_saturation_three
#print axioms Article77.q_saturation_robust_cap
#print axioms Article77.q_saturation_strict
#print axioms Article77.q_nominal_mu_exact
```

```
#print axioms Article77.q_nominal_mu_positive
#print axioms Article77.q_nominal_availability_unit
#print axioms Article77.q_nominal_availability_lower
#print axioms Article77.q_nominal_flux_bounds
#print axioms Article77.q_nominal_flux_lower
#print axioms Article77.q_nominal_flux_strict_feedback
#print axioms Article77.q_nominal_speed_lower
#print axioms Article77.q_nominal_speed_positive
#print axioms Article77.q_nominal_crossing_constants
#print axioms Article77.q_nominal_margin_speed
#print axioms Article77.q_nominal_margin_constants
#print axioms Article77.q_nominal_material_ledgers
#print axioms Article77.q_nominal_operating_ledger_bounds
#print axioms Article77.q_nominal_global_ledger_bounds
#print axioms Article77.q_nominal_energy_floor
#print axioms Article77.q_nominal_water_geometry_constants
#print axioms Article77.q_nominal_vectorfield_lower
#print axioms Article77.q_nominal_vectorfield_speed
#print axioms Article77.q_robust_switch_multiplier
#print axioms Article77.q_robust_mu_exact
#print axioms Article77.q_robust_availability_lower
#print axioms Article77.q_robust_speed_lower
#print axioms Article77.q_robust_speed_positive
#print axioms Article77.q_robust_crossing_constants
#print axioms Article77.q_robust_flux_cap_constants
#print axioms Article77.q_robust_material_bounds
#print axioms Article77.q_robust_area_water_geometry_constants
#print axioms Article77.q_robust_ablation_energy_constants
#print axioms Article77.q_robust_energy_floor
#print axioms Article77.q_terminal_margin_transfer
#print axioms Article77.q_fractional_count_error
#print axioms Article77.q_inverse_scale_margin
#print axioms Article77.q_count_threshold_transfer
#print axioms Article77.q_no_productive_reserve_return
#print axioms Article77.q_no_positive_counter_return
#print axioms Article77.q_normalized_daughter_areas
#print axioms Article77.q_proportional_allocation_conservation
#print axioms Article77.q_lumen_concentration_preserved
#print axioms Article77.q_charge_density_share_iff
#print axioms Article77.q_asymmetric_charge_density_example
#print axioms Article77.q_return_slope_interval
#print axioms Article77.q_negative_slope_counterexample
#print axioms Article77.q_robust_selfmap_margin
#print axioms Article77.q_robust_contraction_margin
#print axioms Article77.q_donnan_rationalized_bound
#print axioms Article77.q_reserve_ratio_unit
#print axioms Article77.q_robust_availability_upper
#print axioms Article77.q_robust_flux_upper
#print axioms Article77.q_robust_vectorfield_lower
#print axioms Article77.q_robust_vectorfield_speed
```

Actual per-theorem dependency audit

The following is the successful compiler output for every named theorem, including the elementary rational helper theorems. Only the standard foundational axioms described above may occur. There are no custom theorem axioms, proof holes, native decision procedures or unsafe declarations.

```
'Article77.execution_composition' does not depend on any axioms
'Article77.execution_invariant' does not depend on any axioms
'Article77.copy_prefix_injective' depends on axioms: [propext]
'Article77.different_admissions_disjoint' does not depend on any axioms
'Article77.owner_partition_exhaustive' depends on axioms: [propext]
'Article77.owner_partition_disjoint' does not depend on any axioms
'Article77.moved_support_owns' depends on axioms: [propext]
'Article77.moved_support_frame' depends on axioms: [propext]
```

```
'Article77.different_owner_updates_commute' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.initial_material_safe' depends on axioms: [propext, Quot.sound]
'Article77.material_step_preserves' depends on axioms: [propext, Quot.sound]
'Article77.all_material_runs_preserve_ancestry' depends on axioms: [propext, Quot.sound]
'Article77.released_product_origins_unique' depends on axioms: [propext, Quot.sound]
'Article77.material_admitted_monotone' depends on axioms: [propext, Quot.sound]
'Article77.old_origins_never_deleted' depends on axioms: [propext, Quot.sound]
'Article77.initial_typed_material_safe' depends on axioms: [propext, Quot.sound]
'Article77.typed_material_step_preserves' depends on axioms: [propext, Quot.sound]
'Article77.every_release_is_nonempty_and_traced' depends on axioms: [propext, Quot.sound]
'Article77.allocation_cannot_release' depends on axioms: [propext]
'Article77.released_cannot_release_again' depends on axioms: [propext]
'Article77.disjoint_writes_can_disable_guard' does not depend on any axioms
'Article77.five_neutral_precursor_formulas' does not depend on any axioms
'Article77.substitution_hydrogen_balance' depends on axioms: [propext, Quot.sound]
'Article77.reservation_has_38_constituents' does not depend on any axioms
'Article77.each_reserved_support_unique' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.reserved_origin_parent' depends on axioms: [propext, Quot.sound]
'Article77.reservation_parent_names_distinct' does not depend on any axioms
'Article77.reservation_has_810_distinct_carbons' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.reservation_role_counts' does not depend on any axioms
'Article77.reservation_parent_role_correct' does not depend on any axioms
'Article77.role_origin_has_unique_parent_role' depends on axioms: [propext, Quot.sound]
'Article77.reservation_role_supports_disjoint' depends on axioms: [propext, Quot.sound]
'Article77.one_event_open_balance' depends on axioms: [propext, Quot.sound]
'Article77.arbitrary_trace_open_balance' depends on axioms: [propext, Quot.sound]
'Article77.exact_open_balance' depends on axioms: [propext, Quot.sound]
'Article77.outside_admission_no_inside_change' does not depend on any axioms
'Article77.export_return_counts_once_each' depends on axioms: [propext]
'Article77.integer_block_carbon_identity' depends on axioms: [propext, Quot.sound]
'Article77.production_preserves_ledgers' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.deactivation_preserves_ledgers' depends on axioms: [propext, Quot.sound]
'Article77.count_step_carbon_preserved' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.all_count_runs_preserve_carbon' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.event_count_ledgers' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.event_count_carrier_floor' depends on axioms: [propext, Quot.sound]
'Article77.event_count_active_plus_inactive' depends on axioms: [propext, Quot.sound]
'Article77.event_count_total_bound' depends on axioms: [propext, Quot.sound]
'Article77.exhausted_donor_blocks' depends on axioms: [propext, Quot.sound]
'Article77.event_production_identity' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.event_deactivation_identity' depends on axioms: [propext, Quot.sound]
'Article77.all_production_blocks_exist' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.available_deactivations_exist' depends on axioms: [propext, Quot.sound]
'Article77.every_bounded_count_state_reachable' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.concatenated_cost' depends on axioms: [propext]
'Article77.prefix_withdrawal_budget' depends on axioms: [propext, Quot.sound]
'Article77.bounded_cycle_cost' depends on axioms: [propext, Quot.sound]
'Article77.finite_cycle_reserve_floor' depends on axioms: [propext, Quot.sound]
'Article77.binary_tree_parent_count' depends on axioms: [propext, Quot.sound]
'Article77.binary_tree_cost' depends on axioms: [propext, Quot.sound]
'Article77.daughter_minima_necessary' depends on axioms: [propext, Quot.sound]
'Article77.daughter_minima_sufficient' depends on axioms: [propext, Quot.sound]
'Article77.one_indivisible_pore_insufficient' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.integer_partition_carbon' depends on axioms: [propext]
'Article77.complement_involutive' does not depend on any axioms
'Article77.complement_map_twice' depends on axioms: [propext]
'Article77.reverse_complement_involutive' depends on axioms: [propext]
'Article77.reverse_complement_length' depends on axioms: [propext]
'Article77.qDivLe' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.qLeDiv' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.qMulLe' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.qUnitProduct' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_saturation_bounds' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_saturation_three' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_saturation_robust_cap' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_saturation_strict' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_mu_exact' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_mu_positive' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_availability_unit' depends on axioms: [propext, Classical.choice, Quot.sound]
```

```
'Article77.q_nominal_availability_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_flux_bounds' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_flux_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_flux_strict_feedback' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_speed_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_speed_positive' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_crossing_constants' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_margin_speed' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_margin_constants' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_material_ledgers' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_operating_ledger_bounds' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_global_ledger_bounds' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_energy_floor' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_water_geometry_constants' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_vectorfield_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_nominal_vectorfield_speed' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_switch_multiplier' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_mu_exact' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_availability_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_speed_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_speed_positive' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_crossing_constants' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_flux_cap_constants' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_material_bounds' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_area_water_geometry_constants' depends on axioms: [propext, Classical.choice, Quot.sound
    ]
'Article77.q_robust_ablation_energy_constants' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_energy_floor' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_terminal_margin_transfer' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_fractional_count_error' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_inverse_scale_margin' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_count_threshold_transfer' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_no_productive_reserve_return' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_no_positive_counter_return' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_normalized_daughter_areas' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_proportional_allocation_conservation' depends on axioms: [propext, Classical.choice, Quot.sound
    ]
'Article77.q_lumen_concentration_preserved' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_charge_density_share_iff' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_asymmetric_charge_density_example' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_return_slope_interval' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_negative_slope_counterexample' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_selfmap_margin' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_contraction_margin' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_donnan_rationalized_bound' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_reserve_ratio_unit' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_availability_upper' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_flux_upper' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_vectorfield_lower' depends on axioms: [propext, Classical.choice, Quot.sound]
'Article77.q_robust_vectorfield_speed' depends on axioms: [propext, Classical.choice, Quot.sound]
```

Publication description

Supported description: “A partial finite/integer/rational Lean 4 formalization of Article 77, containing 125 kernel-checked theorems, complete source, transitive dependency audit and an explicit coverage map.” This description is deliberately narrower than a full-manuscript, real-ODE, physical-synthesis or molecular-realization certificate.