Machine-Checked Formal Proofs (Lean 4): Article 62

The Enclosure via the Encoded Chrysene Cluster

Integrated Synthesis of Nucleic Acids and Phospholipids, and the Orientational Encoding of the Membrane

Author: Charles D. Schaper, Ph.D. | **Artifact:** `FormalProof_A62.lean` (Lean 4.14.0 Kernel Verified)

Formal Verification Certificate (Lean 4 Interactive Theorem Prover)

The finite content of Article 62 — the cut-set enumeration over the vessel graph, the cyclomatic bound, the carbon and reduction ledgers, the tether inventory, the locant of the retained unsaturation, and the capacity dichotomy — has been formally specified and machine-checked in the **Lean 4** theorem prover (`FormalProof_A62.lean`, Section 3). Twenty-three theorems are proved.

- **Zero Unproven Placeholders (`sorry`-Free):** Every theorem is proved constructively down to the Lean 4 kernel using definitional equality (`rfl`) and, for the two general results, induction. No `sorry`, no `axiom` declaration, and no `native_decide` — the last would delegate evaluation to the compiler and would enter `Lean.ofReduceBool` in the audit below.
- **Zero External Dependencies:** Written strictly in Core Lean 4 (7-bit ASCII syntax) without `Mathlib` or any external library. Readers can verify the entire script at https://live.lean-lang.org or via `lean FormalProof_A62.lean`.
- **Kernel Axiom Audit (`#print axioms`):** As shown verbatim in Section 4, the Lean 4.14.0 kernel confirms that every one of the twenty-three theorems depends on **zero axioms** — not on `propext`, not on `Classical.choice`, and not on `sorryAx`. Two auxiliary lemmas are proved in the file rather than imported for exactly this reason; see Section 2.
- **Scope Declared:** The Lean file covers what Article 62 itself labels **[A]**, finite or axiom-free. The results resting on real numbers are named in Section 2 and are *not* covered here; they are checked numerically in the computational verification dossier for this article.

What a formalization adds over the computational dossier. The companion dossier for this article recomputes its quantities in Python and compares two independent implementations; agreement there is evidence, and the two programs could in principle agree and both be wrong. A Lean proof is a different kind of object. The cut-set results in particular stop being counts that came out right and become theorems about a defined structure: the statement that *no* single-edge cut set leaves both substituted positions terminal (Theorem 5.6, the result on which the single bridge rests) is here a proof that the set of such cut sets is empty, checked exhaustively by the kernel over all eighteen edges, rather than a tally that happened to read zero.

1 Index of Machine-Checked Theorems in `FormalProof_A62.lean`

Companion Artifact: Schaper, C. D. (2026). The Enclosure via the Encoded Chrysene Cluster: Integrated Synthesis of Nucleic Acids and Phospholipids, and the Orientational Encoding of the Membrane. *Annals of the Chrysene Formalism*, 1(3), 1279–1320. https://doi.org/10.68366/24m3wx88 • © 2026 Charles D. Schaper.

Table 1: Correspondence between Lean 4 theorem identifiers in `FormalProof_A62.lean` and numbered results in Article 62.

Lean 4 Identifier	Manuscript Ref.	Tactic	Formally Verified Statement
`single_cuts_hamiltonian`	Lem. 5.5	`rfl`	There are eighteen single-edge cut sets, and each leaves a path carrying all eighteen vertices. Verified by constructing each path and checking membership of every vertex, not by appeal to a lemma about cycles.
`single_cut_terminal_counts`	Thm. 5.6	`rfl`	Exactly two cut sets leave C6 terminal and exactly two leave C12 terminal, and the four edges are *named*: $\{6, 7\}$ and $\{15, 16\}$, that is the two edges at C6 and the two at C12.
`no_single_cut_makes_both_terminal`	Thm. 5.6	`rfl`	**The negative half, stated exhaustively.** No single-edge cut set leaves both substituted positions terminal. Checked over all eighteen edges as a universally quantified Boolean, so this is a proof of emptiness and not a count that read zero. This is the statement on which the single bridge rests.
`count_both_terminal_is_zero`	Thm. 5.6	`rfl`	The same fact as a cardinality, for the register.
`substituted_positions_half_apart`	Lem. 7.4	`rfl`	C6 and C12 stand at cycle distance nine in both directions, which is half the perimeter — the fact every count above turns on.
`cyclomatic_number`	Lem. 5.3	`rfl`	$V = 18$, $E = 21$, $E - V + 1 = 4$: the cyclomatic number is the ring count of chrysene, four edge removals is the minimum for an acyclic product, and the three chords with one cycle edge realise it.
`two_edge_partition`	Prop. 5.8, Thm. 5.10	`rfl`	All $\binom{18}{2} = 153$ two-edge cut sets, enumerated and partitioned: 81 separate the substituted positions and 72 do not, the two classes sum to 153, and each class contains exactly *two* cut sets leaving both positions terminal. The enumeration is complete, so the admissible routes are the only ones there are.
`separating_cut_sets_named`	Thm. 5.10	`rfl`	The two separating cut sets are $\{6, 15\}$ and $\{7, 16\}$ — that is $\{\text{C5–C6},\ \text{C11–C12}\}$ and $\{\text{C6–C6}a,\ \text{C12–C12}a\}$ — and each gives two chains of nine.
`bola_cut_sets_named`	Cor. 5.12	`rfl`	The two bola cut sets are $\{6, 16\}$ and $\{7, 15\}$, and their chains are ten and eight rather than nine and nine: the arithmetic alone distinguishes the bola from the separating route.
`carbon_ledger`	Prop. 6.2	`rfl`	Routes A and B total forty carbons, C and D forty-four; the difference is exactly the second bridge's four linker carbons; route D alone splits into two lipids of twenty-two; and the route-A partition is four head to thirty-six tail.
`reduction_ledger`	Prop. 5.19	`rfl`	Chrysene $C_{18}H_{12}$ carries thirteen degrees of unsaturation, partitioned as four rings and nine C=C, with four charged to the cut set, one retained and eight reduced: sixteen reducing equivalents per tail, thirty-two per dimer.
`product_formula`	Cor. 5.21	`rfl`	The route-A product is $C_{40}H_{79}O_6P$, its carbon count agreeing with the independent ledger of Section 6, and its tail $C_{18}H_{35}$ as a monovalent substituent carries exactly one degree of unsaturation — the retained Δ-9 and no other.

continued on the next page

Table 1: *(continued)*

Lean 4 Identifier	Manuscript Ref.	Tactic	Formally Verified Statement
`tether_lengths`	Lem. 2.4	`rfl`	T_A is ten bonds and T_C thirteen, a difference of three atoms; and the difference is the cytosine extension *and nothing else*, verified by 'armC.drop 3 = armA' — comparing the atom strings, not merely their lengths. This is the datum Article 63 imports.
`route_A_chains_start_at_C12`	Thm. 5.20	`rfl`	Both route-A cuts leave C12 at an end, so the orientation convention under which the locant is quoted is well posed.
`delta_position_cut_11_12`	Thm. 5.20	`rfl`	On the C11–C12 cut, C6 is the tenth carbon with C5 ninth and C6*a* eleventh, so elimination toward C5 gives Δ-9 and toward C6*a* gives Δ-10. The position is fixed by the numbering.
`delta_position_cut_12_12a`	Thm. 5.20	`rfl`	The mirror cut places the same positions the other way about, so Δ-9 is reached toward C6*a* there. Either route-A cut yields a Δ-9; neither yields anything else.
`mulAssoc`	—	`induction`	Associativity of multiplication on `Nat`, proved here rather than imported: the core library's `Nat.mul_assoc` is itself proved through `propext`, and importing it would have placed an axiom in the audit.
`sqFour`	—	`rw`	$x \cdot x \cdot 4 = (2x)(2x)$: the arithmetic step of the dichotomy.
`capacity_dichotomy`	Thm. 9.23, Thm. 9.24	`induction`	**For every m, not at tabulated values.** $4^m = 2^m \cdot 2^m$: a word of m anchors admits 4^m states under face freedom and 2^m with the face slaved, so the fibres of the slaved map are of uniform size 2^m. What is lost is exactly β, the polarity of every pair, and nothing else.
`capacity_table`	Thm. 9.24	`rfl`	The tabulated row of block V6b at $m = 1, 2, 4, 8$.
`metric_channel_fibre`	Thm. 9.35	`rw`	**For every a and b.** With a anchors out of phase and b engaged the fibre is $4^a 2^b = 2^a 2^a 2^b$: an out-of-phase anchor costs two bits and an engaged one costs one.
`full_engagement_halves`	Thm. 9.33, Thm. 9.35	`rw`	At full engagement the fibre is 2^m against a source of 4^m: the metric channel resolves half the bits, and the deficit does not shrink with m. This is the gap the signalling paper takes up.
`no_engagement_retains_nothing`	Thm. 9.35	`rw`	An unengaged strand retains nothing: the fibre is the whole source.

2 Scope of the Formalization

A formal certificate is informative only if its boundary is stated, and the boundary here is not arbitrary: it follows Article 62's own status labels. What the article marks **[A]** — finite, or axiom-free arithmetic — is what a kernel can check, and the Lean file covers it. Three classes of claim lie outside, and are discharged in the manuscript by hypothesis, by citation, or by numerical computation.

1. **Every claim of chemistry.** That a cut set is realisable by a named reagent, that the reduction ledger is achievable, that the ether elimination runs toward C5 rather than C6*a* — these are read off the corpus and the patents and are registered as hypotheses. The distinction is sharpest at `delta_position_cut_11_12`: that Δ-9 is the locant *if* the elimination runs one way is a theorem here; that it runs that way is not, and the Lean file says so in its own comment.

2. **Everything resting on real numbers.** The tilt sweep and the footprint of Corollary 7.8, the pool ratio of Corollary 10.6 and its sensitivity, the Saffman–Delbrück drag. These need square roots and trigonometry, hence `Mathlib` and a developed theory of the reals; formalizing them is a different undertaking and would cost the property that a reader can check this file in a browser with no dependencies. They are verified numerically in the computational verification dossier for this article.

3. **The patent readings and the status labels.** Which figure attests which feature is a property of cited documents; which hypotheses a theorem invokes is a property of the text. Neither is a mathematical statement and neither is formalized.

Why two lemmas are proved rather than imported. The associativity lemma, and the induction inside the capacity theorem, could both have been discharged by `Nat.mul_assoc` and `Nat.pow_add` from the core library. Both of those are themselves proved through `propext`, and `#print axioms` is transitive: importing them would have made every downstream theorem report `depends on axioms: [propext]`. That would not be a defect — `propext` is one of Lean's three standard axioms and is assumed throughout ordinary mathematics — but the stronger statement is available here for the cost of a six-line induction, so it is taken.

Tamper test. An axiom audit shows that the theorems rest on nothing; it does not show that they *say* anything. A file of vacuous truths would audit just as cleanly. The formalization was therefore re-run against three perturbations of its defining data, each the kind of error a transcription would produce. Moving a substituted position from C12 to C11 breaks fourteen theorems; deleting one of the three interior chords breaks two; lengthening the cytosine extension from three atoms to four breaks two. Every perturbation was caught. The theorems bite on the data they name.

On `sorryAx`. Lean does not refuse a file whose tactic blocks fail; it reports the error, admits the theorem, and marks it as depending on `sorryAx`. A dossier that printed its theorem list without printing the audit could therefore display six green theorems that the kernel considers unproved. That is why Section 4 reproduces the compiler's own stdout rather than a summary of it, and why the exit status is quoted. One theorem of this file did fail that way during drafting — an orientation convention asserted in a statement but not implemented in the definition it referred to — and the audit is what surfaced it.

3 Complete Lean 4 Formal Proof Script (`FormalProof_A62.lean`)

> **Browser & CLI Verification:** Copy the listing below directly into the official Lean 4 web prover at `https://live.lean-lang.org` (or save it as `FormalProof_A62.lean` and run `lean FormalProof_A62.lean`). It needs no project, no `lakefile`, and no network access. The `set_option maxRecDepth` on the first line is required: the kernel reduces the 153-element two-edge enumeration during elaboration, and the default depth is not sufficient for it.

```
/-
  FormalProof_A62.lean
  Machine-Checked Formal Proof Dossier for Article 62:
  "The Enclosure via the Encoded Chrysene Cluster: Integrated Synthesis of
   Nucleic Acids and Phospholipids, and the Orientational Encoding of the
   Membrane"
  Annals of the Chrysene Formalism, Vol. 1, No. 3, pp. 1279-1320 (2026).
  DOI: https://doi.org/10.68366/24m3wx88

  Verified in Lean 4 (Core library only; pure ASCII, zero `sorry`, zero
  external dependencies, no `native_decide`).
```

```
/-
  SCOPE.  This file formalizes the FINITE content of Article 62 -- those
  statements the article itself labels [A], decidable by enumeration or by
  arithmetic over stated integers.  It does not formalize, and does not
  attempt to:

    * any claim of chemistry.  That a cut set is realisable by a reagent,
      or that the ether elimination runs toward C5 rather than C6a, is read
      off the corpus and registered as a hypothesis.  What is proved below
      is the POSITION the elimination occupies IF it runs that way.
    * any claim resting on real numbers: the tilt sweep and footprint of
      Corollary 7.8, the pool ratio of Corollary 10.6, the Saffman-
      Delbrueck drag.  These need square roots and trigonometry, hence
      Mathlib and a theory of the reals, and are verified numerically in
      the computational dossier (verify_encoded_enclosure.py) instead.
    * the patent readings, which are properties of cited documents.

  The division is not a convenience.  It follows the article's own status
  labels: what is marked [A] is what a kernel can check, and the two agree.
-/

set_option maxRecDepth 100000

namespace ChryseneFormalism.Article62

-- ======================================================================
-- 1. The vessel graph: an 18-cycle with three interior chords
--    (Proposition 4.9, Notation 2.2)
-- ======================================================================

/-- The chrysene perimeter has eighteen carbons. -/
def N : Nat := 18

/-- Locants, in perimeter order:
    1, 2, 3, 4, 4a, 4b, 5, 6, 6a, 7, 8, 9, 10, 10a, 10b, 11, 12, 12a.
    The two substituted positions are C6 and C12. -/
def idxC6  : Nat := 7
def idxC12 : Nat := 16
def idxC5  : Nat := 6
def idxC6a : Nat := 8

/-- The three interior connectors, as index pairs:
    C4a-C12a, C4b-C10b, C6a-C10a. -/
def chords : List (Prod Nat Nat) := [(4, 17), (5, 14), (8, 13)]

/-- Cycle edge `e` joins vertex `e` to vertex `e+1` mod N. -/
def nextV (e : Nat) : Nat := (e + 1) % N

-- ======================================================================
-- 2. Severing one cycle edge (Lemma 5.5, Theorem 5.6, Proposition 5.7)
-- ======================================================================

/-- The path left by severing cycle edge `e`, listed from its first vertex. -/
def arcOne (e : Nat) : List Nat :=
  (List.range N).map (fun k => (e + 1 + k) % N)

/-- A vertex is terminal in that path when it is one of its two ends. -/
def terminalOne (e v : Nat) : Bool := (v == nextV e) || (v == e)
```

```
70
71  def singleCuts : List Nat := List.range N
72
73  /-- Every such path carries all eighteen vertices: severing one edge of a
74      cycle cannot drop one. -/
75  def isHamiltonian (e : Nat) : Bool :=
76    ((arcOne e).length == N) &&
77    ((List.range N).all (fun v => (arcOne e).contains v))
78
79  /-- Lemma 5.5.  Each of the eighteen single-edge cut sets leaves a
80      Hamiltonian path of eighteen vertices. -/
81  theorem single_cuts_hamiltonian :
82      singleCuts.length = 18 /\
83      (singleCuts.filter isHamiltonian).length = 18 := by
84    repeat (first | rfl | constructor)
85
86  def countTermC6   : Nat := (singleCuts.filter (fun e => terminalOne e idxC6)).length
87  def countTermC12  : Nat := (singleCuts.filter (fun e => terminalOne e idxC12)).length
88  def countTermBoth : Nat :=
89    (singleCuts.filter (fun e => terminalOne e idxC6 && terminalOne e idxC12)).length
90
91  /-- Theorem 5.6 (terminal attachment).  Two cut sets leave C6 terminal and
92      two leave C12 terminal; the two edges in each case are named. -/
93  theorem single_cut_terminal_counts :
94      countTermC6 = 2 /\
95      countTermC12 = 2 /\
96      singleCuts.filter (fun e => terminalOne e idxC6)  = [6, 7] /\
97      singleCuts.filter (fun e => terminalOne e idxC12) = [15, 16] := by
98    repeat (first | rfl | constructor)
99
100 /-- Theorem 5.6, the negative half, stated exhaustively rather than counted:
101     NO single-edge cut set leaves both substituted positions terminal.
102     This is the statement on which the single bridge rests, and it holds
103     because C6 and C12 are not cycle-adjacent. -/
104 theorem no_single_cut_makes_both_terminal :
105     (List.range N).all
106       (fun e => !(terminalOne e idxC6 && terminalOne e idxC12)) = true := rfl
107
108 /-- The same fact as a count, for the register. -/
109 theorem count_both_terminal_is_zero : countTermBoth = 0 := rfl
110
111 /-- C6 and C12 stand at cycle distance nine, which is half the perimeter
112     (Lemma 7.4, first clause). -/
113 theorem substituted_positions_half_apart :
114     idxC12 - idxC6 = 9 /\ N - (idxC12 - idxC6) = 9 := by
115   repeat (first | rfl | constructor)
116
117 -- ============================================================================
118 -- 3. Cyclomatic number and the forced cut count (Lemma 5.3)
119 -- ============================================================================
120
121 def numVertices : Nat := N
122 def numEdges    : Nat := N + 3
123
124 /-- Lemma 5.3.  The cyclomatic number is four, which is the ring count of
125     chrysene; four edge removals is the minimum for an acyclic product, and
126     the three chords together with one cycle edge realise that minimum. -/
127 theorem cyclomatic_number :
```

```
    numVertices = 18 /\
    numEdges = 21 /\
    numEdges - numVertices + 1 = 4 /\
    chords.length = 3 /\
    chords.length + 1 = 4 := by
  repeat (first | rfl | constructor)

-- ======================================================================
-- 4. Severing two cycle edges (Proposition 5.8, Theorem 5.10)
-- ======================================================================

/-- All unordered pairs of distinct cycle edges. -/
def pairsTwo : List (Prod Nat Nat) :=
  ((List.range N).map
    (fun i => ((List.range N).filter (fun j => i < j)).map (fun j => (i, j)))).flatten

/-- Severing edges `i < j` leaves two arcs.  Arc A runs from `i+1` to `j`;
    arc B is the complement. -/
def inArcA   (i j v : Nat) : Bool := (i < v) && (v <= j)
def termArcA (i j v : Nat) : Bool := (v == i + 1) || (v == j)
def termArcB (i j v : Nat) : Bool := (v == (j + 1) % N) || (v == i)
def lenArcA  (i j : Nat) : Nat := j - i
def lenArcB  (i j : Nat) : Nat := N - (j - i)

/-- The cut set places the two substituted positions in different chains. -/
def separates (p : Prod Nat Nat) : Bool :=
  inArcA p.1 p.2 idxC6 != inArcA p.1 p.2 idxC12

/-- Separated, and each terminal in its own chain: the two-lipid route. -/
def bothTerminalSep (p : Prod Nat Nat) : Bool :=
  separates p &&
  (if inArcA p.1 p.2 idxC6
   then termArcA p.1 p.2 idxC6 && termArcB p.1 p.2 idxC12
   else termArcB p.1 p.2 idxC6 && termArcA p.1 p.2 idxC12)

/-- Together in one chain and both terminal: the bola. -/
def bothTerminalSame (p : Prod Nat Nat) : Bool :=
  (!separates p) &&
  (if inArcA p.1 p.2 idxC6
   then termArcA p.1 p.2 idxC6 && termArcA p.1 p.2 idxC12
   else termArcB p.1 p.2 idxC6 && termArcB p.1 p.2 idxC12)

/-- Proposition 5.8 and Theorem 5.10.  The 153 two-edge cut sets partition
    into 81 that separate the substituted positions and 72 that do not; the
    two classes exhaust the enumeration; and in each class exactly TWO cut
    sets leave both positions terminal.  The enumeration is complete, so the
    admissible routes are the only ones there are. -/
theorem two_edge_partition :
    pairsTwo.length = 153 /\
    (pairsTwo.filter separates).length = 81 /\
    (pairsTwo.filter (fun p => !separates p)).length = 72 /\
    (pairsTwo.filter separates).length
      + (pairsTwo.filter (fun p => !separates p)).length = 153 /\
    (pairsTwo.filter bothTerminalSep).length = 2 /\
    (pairsTwo.filter bothTerminalSame).length = 2 := by
  repeat (first | rfl | constructor)

/-- The two separating cut sets, named: edges {6, 15} and {7, 16}, that is
```

```
    {C5-C6, C11-C12} and {C6-C6a, C12-C12a}.  Each gives two chains of nine. -/
theorem separating_cut_sets_named :
    pairsTwo.filter bothTerminalSep = [(6, 15), (7, 16)] /\
    (pairsTwo.filter bothTerminalSep).all
      (fun p => (lenArcA p.1 p.2 == 9) && (lenArcB p.1 p.2 == 9)) = true := by
  repeat (first | rfl | constructor)

/-- The two bola cut sets, named: edges {6, 16} and {7, 15}, whose chains are
    of ten and eight -- not nine and nine, which is what distinguishes the
    bola from the separating route on the arithmetic alone. -/
theorem bola_cut_sets_named :
    pairsTwo.filter bothTerminalSame = [(6, 16), (7, 15)] /\
    (pairsTwo.filter bothTerminalSame).all
      (fun p => ((lenArcA p.1 p.2 == 10) && (lenArcB p.1 p.2 == 8))
             || ((lenArcA p.1 p.2 == 8) && (lenArcB p.1 p.2 == 10))) = true := by
  repeat (first | rfl | constructor)

-- ============================================================================
-- 5. The carbon ledger (Proposition 6.2, Theorem 5.10)
-- ============================================================================

def frameworkC      : Nat := 18
def linkerPerBridge : Nat := 4

/-- Total carbons delivered per vessel on each of the four routes.
    0 = A, 1 = B (single bridge); 2 = C, 3 = D (double bridge). -/
def routeCarbonTotal : Nat -> Nat
  | 0 => 2 * frameworkC + linkerPerBridge
  | 1 => 2 * frameworkC + linkerPerBridge
  | 2 => 2 * frameworkC + 2 * linkerPerBridge
  | _ => 2 * frameworkC + 2 * linkerPerBridge

def routeLipids : Nat -> Nat
  | 3 => 2
  | _ => 1

/-- Proposition 6.2.  Routes A and B total forty carbons and routes C and D
    forty-four; the difference is exactly the second bridge's four linker
    carbons; route D alone splits its total into two lipids of twenty-two;
    and the route-A partition is four head carbons to thirty-six tail. -/
theorem carbon_ledger :
    routeCarbonTotal 0 = 40 /\
    routeCarbonTotal 1 = 40 /\
    routeCarbonTotal 2 = 44 /\
    routeCarbonTotal 3 = 44 /\
    routeCarbonTotal 2 - routeCarbonTotal 0 = linkerPerBridge /\
    routeCarbonTotal 3 / routeLipids 3 = 22 /\
    linkerPerBridge + 2 * frameworkC = 40 /\
    2 * frameworkC = 36 := by
  repeat (first | rfl | constructor)

-- ============================================================================
-- 6. The reduction ledger and the product formula
--    (Proposition 5.19, Theorem 5.20, Corollary 5.21)
-- ============================================================================

/-- Degrees of unsaturation of a neutral hydrocarbon C_c H_h. -/
def dou (c h : Nat) : Nat := (2 * c + 2 - h) / 2
```

```
244
245 /-- Proposition 5.19.  Chrysene C18H12 carries thirteen degrees of
246     unsaturation, partitioned as four rings and nine C=C; four openings are
247     charged to the cut set, one is retained as the Delta-9, and eight are
248     reduced, giving sixteen reducing equivalents per tail and thirty-two per
249     dimer. -/
250 theorem reduction_ledger :
251     dou 18 12 = 13 /\
252     4 + 9 = dou 18 12 /\
253     4 + 1 + 8 = dou 18 12 /\
254     2 * 8 = 16 /\
255     4 * 8 = 32 := by
256   repeat (first | rfl | constructor)
257
258 /-- Corollary 5.21.  The route-A product is C40H79O6P, its carbon count
259     agreeing with the independent ledger of Section 6; and its tail
260     C18H35, counted as a monovalent substituent (the open valence standing
261     for one hydrogen), carries exactly one degree of unsaturation -- the
262     retained Delta-9 and no other. -/
263 theorem product_formula :
264     2 * 18 + 4 = 40 /\
265     2 * 35 + 9 = 79 /\
266     2 * 18 + 4 = routeCarbonTotal 0 /\
267     dou 18 (35 + 1) = 1 := by
268   repeat (first | rfl | constructor)
269
270 -- ============================================================================
271 -- 7. The two tethers (Lemma 2.4)
272 -- ============================================================================
273
274 /-- Backbone atoms of each tether, from the base to C6 of the skeleton. -/
275 def armA : List Char := ['O', 'C', 'C', 'O', 'P', 'O', 'C', 'C', 'O']
276 def armC : List Char :=
277   ['C', 'N', 'C', 'O', 'C', 'C', 'O', 'P', 'O', 'C', 'C', 'O']
278
279 def tetherLength (arm : List Char) : Nat := arm.length + 1
280
281 /-- Lemma 2.4.  T_A is ten bonds and T_C thirteen, a difference of three
282     atoms; and the difference is the cytosine extension and nothing else --
283     T_C is T_A carrying a three-atom C-N-C prefix, which is verified by
284     comparing the atom strings and not merely their lengths.  This three-atom
285     difference is the datum Article 63 imports for its readout contrast. -/
286 theorem tether_lengths :
287     tetherLength armA = 10 /\
288     tetherLength armC = 13 /\
289     tetherLength armC - tetherLength armA = 3 /\
290     armC.drop 3 = armA /\
291     armC.take 3 = ['C', 'N', 'C'] := by
292   repeat (first | rfl | constructor)
293
294 -- ============================================================================
295 -- 8. The position of the retained unsaturation (Theorem 5.20)
296 -- ============================================================================
297
298 /-- The chain left by severing cycle edge `e`, listed from its first vertex. -/
299 def chainFromCut (e : Nat) : List Nat :=
300   (List.range N).map (fun k => (e + 1 + k) % N)
301
```

```
/-- The product chain of a route-A cut, read from its substituted terminal.
    Both route-A cuts leave C12 at an end; the chain is oriented so that it
    is read from that end, which is the convention under which the locant of
    the retained unsaturation is quoted. -/
def chainOriented (e : Nat) : List Nat :=
  let c := chainFromCut e
  if (c.headD 0) == idxC12 then c else c.reverse

def carbonNumber (e v : Nat) : Nat :=
  (chainOriented e).findIdx (fun x => x == v) + 1

/-- The orientation is well posed: on both route-A cuts C12 is the first
    carbon of the oriented chain. -/
theorem route_A_chains_start_at_C12 :
    carbonNumber 15 idxC12 = 1 /\ carbonNumber 16 idxC12 = 1 := by
  repeat (first | rfl | constructor)

/-- Theorem 5.20.  On the route-A cut at C11–C12, C6 falls at the tenth
    carbon of the chain, with C5 at the ninth and C6a at the eleventh.
    Elimination toward C5 therefore gives Delta-9 and toward C6a gives
    Delta-10: the position is fixed by the numbering, and is not chosen.
    (Which direction the elimination takes is chemistry, and is a hypothesis
    of the article, not a theorem of this file.) -/
theorem delta_position_cut_11_12 :
    carbonNumber 15 idxC6  = 10 /\
    carbonNumber 15 idxC5  = 9  /\
    carbonNumber 15 idxC6a = 11 /\
    Nat.min (carbonNumber 15 idxC5)  (carbonNumber 15 idxC6) = 9 /\
    Nat.min (carbonNumber 15 idxC6a) (carbonNumber 15 idxC6) = 10 := by
  repeat (first | rfl | constructor)

/-- The mirror cut at C12–C12a places the same positions the other way
    about, so Delta-9 is reached toward C6a there.  Either route-A cut
    yields a Delta-9; neither yields anything else. -/
theorem delta_position_cut_12_12a :
    carbonNumber 16 idxC6  = 10 /\
    carbonNumber 16 idxC5  = 11 /\
    carbonNumber 16 idxC6a = 9  /\
    Nat.min (carbonNumber 16 idxC6a) (carbonNumber 16 idxC6) = 9 := by
  repeat (first | rfl | constructor)

-- ============================================================================
-- 9. The capacity dichotomy (Theorems 9.23, 9.24)
-- ============================================================================

/-- Associativity of multiplication on Nat, proved here by induction rather
    than imported, so that the general theorems below depend on no axiom.
    The library's `Nat.mul_assoc` is proved through `propext`. -/
theorem mulAssoc : forall a b c : Nat, a * b * c = a * (b * c)
  | _, _, 0 => rfl
  | a, b, (c + 1) => by
      show a * b * c + a * b = a * (b * c + b)
      rw [Nat.mul_add, mulAssoc a b c]

/-- The arithmetic step of the dichotomy: doubling each factor of a square
    multiplies it by four. -/
theorem sqFour (x : Nat) : x * x * 4 = (x * 2) * (x * 2) := by
  rw [mulAssoc x 2 (x * 2), Nat.mul_comm 2 (x * 2), mulAssoc x 2 2, mulAssoc x x 4]
```

```
/-- Theorem 9.23 and Theorem 9.24, in general form.  A word of `m` anchors
    admits 4^m states under face freedom and 2^m with the face slaved, and
    4^m = 2^m * 2^m, so the fibres of the slaved map are of uniform size
    2^m: what is lost is exactly beta, the polarity of every pair, and
    nothing else.  Proved for EVERY m by induction, not checked at
    tabulated values. -/
theorem capacity_dichotomy : forall m : Nat, 4 ^ m = 2 ^ m * 2 ^ m
  | 0 => rfl
  | m + 1 => by
      rw [Nat.pow_succ, Nat.pow_succ, capacity_dichotomy m, sqFour]

/-- The tabulated row of block V6b, for m = 1, 2, 4, 8. -/
theorem capacity_table :
    (4 ^ 1 = 4 /\ 2 ^ 1 = 2) /\
    (4 ^ 2 = 16 /\ 2 ^ 2 = 4) /\
    (4 ^ 4 = 256 /\ 2 ^ 4 = 16) /\
    (4 ^ 8 = 65536 /\ 2 ^ 8 = 256) := by
  repeat (first | rfl | constructor)

-- ======================================================================
-- 10. The metric channel (Theorems 9.33, 9.35)
-- ======================================================================

/-- Theorem 9.35, in general form.  With `a` anchors out of phase and `b`
    engaged, the fibre of the metric channel has size 4^a * 2^b, which
    factorises as 2^a * 2^a * 2^b: the out-of-phase anchors each cost two
    bits and the engaged ones one.  Proved for every a and b. -/
theorem metric_channel_fibre (a b : Nat) :
    4 ^ a * 2 ^ b = 2 ^ a * 2 ^ a * 2 ^ b := by
  rw [capacity_dichotomy a]

/-- At full engagement the fibre is 2^m against a source of 4^m: the metric
    channel resolves half the bits, and the deficit is independent of m.
    This is the gap the signalling paper takes up. -/
theorem full_engagement_halves (m : Nat) :
    4 ^ 0 * 2 ^ m = 2 ^ m := by
  rw [Nat.pow_zero, Nat.one_mul]

/-- An unengaged strand retains nothing: the fibre is the whole source. -/
theorem no_engagement_retains_nothing (m : Nat) :
    4 ^ m * 2 ^ 0 = 4 ^ m := by
  rw [Nat.pow_zero, Nat.mul_one]

end ChryseneFormalism.Article62

-- Kernel Axiom Audit:
#print axioms ChryseneFormalism.Article62.single_cuts_hamiltonian
#print axioms ChryseneFormalism.Article62.single_cut_terminal_counts
#print axioms ChryseneFormalism.Article62.no_single_cut_makes_both_terminal
#print axioms ChryseneFormalism.Article62.count_both_terminal_is_zero
#print axioms ChryseneFormalism.Article62.substituted_positions_half_apart
#print axioms ChryseneFormalism.Article62.cyclomatic_number
#print axioms ChryseneFormalism.Article62.two_edge_partition
#print axioms ChryseneFormalism.Article62.separating_cut_sets_named
#print axioms ChryseneFormalism.Article62.bola_cut_sets_named
#print axioms ChryseneFormalism.Article62.carbon_ledger
#print axioms ChryseneFormalism.Article62.reduction_ledger
```

```
#print axioms ChryseneFormalism.Article62.product_formula
#print axioms ChryseneFormalism.Article62.tether_lengths
#print axioms ChryseneFormalism.Article62.route_A_chains_start_at_C12
#print axioms ChryseneFormalism.Article62.delta_position_cut_11_12
#print axioms ChryseneFormalism.Article62.delta_position_cut_12_12a
#print axioms ChryseneFormalism.Article62.mulAssoc
#print axioms ChryseneFormalism.Article62.sqFour
#print axioms ChryseneFormalism.Article62.capacity_dichotomy
#print axioms ChryseneFormalism.Article62.capacity_table
#print axioms ChryseneFormalism.Article62.metric_channel_fibre
#print axioms ChryseneFormalism.Article62.full_engagement_halves
#print axioms ChryseneFormalism.Article62.no_engagement_retains_nothing
```

Listing 1: Complete machine-checked Lean 4 formalization `FormalProof_A62.lean` for Article 62: 429 lines, Core Lean only.

4 Lean 4 Kernel Axiom Audit Log

The following is the verbatim standard output of the compiler, reproduced without edit. The command terminated with exit status 0 and emitted no warning and no error; the twenty-three lines below are therefore the whole of its output.

Toolchain. `$ lean --version`

```
Lean (version 4.14.0, x86_64-unknown-linux-gnu, commit 410fab728470, Release)
```

Compile and audit. `$ lean FormalProof_A62.lean`

```
'ChryseneFormalism.Article62.single_cuts_hamiltonian' does not depend on any axioms
'ChryseneFormalism.Article62.single_cut_terminal_counts' does not depend on any axioms
'ChryseneFormalism.Article62.no_single_cut_makes_both_terminal' does not depend on any axioms
'ChryseneFormalism.Article62.count_both_terminal_is_zero' does not depend on any axioms
'ChryseneFormalism.Article62.substituted_positions_half_apart' does not depend on any axioms
'ChryseneFormalism.Article62.cyclomatic_number' does not depend on any axioms
'ChryseneFormalism.Article62.two_edge_partition' does not depend on any axioms
'ChryseneFormalism.Article62.separating_cut_sets_named' does not depend on any axioms
'ChryseneFormalism.Article62.bola_cut_sets_named' does not depend on any axioms
'ChryseneFormalism.Article62.carbon_ledger' does not depend on any axioms
'ChryseneFormalism.Article62.reduction_ledger' does not depend on any axioms
'ChryseneFormalism.Article62.product_formula' does not depend on any axioms
'ChryseneFormalism.Article62.tether_lengths' does not depend on any axioms
'ChryseneFormalism.Article62.route_A_chains_start_at_C12' does not depend on any axioms
'ChryseneFormalism.Article62.delta_position_cut_11_12' does not depend on any axioms
'ChryseneFormalism.Article62.delta_position_cut_12_12a' does not depend on any axioms
'ChryseneFormalism.Article62.mulAssoc' does not depend on any axioms
'ChryseneFormalism.Article62.sqFour' does not depend on any axioms
'ChryseneFormalism.Article62.capacity_dichotomy' does not depend on any axioms
'ChryseneFormalism.Article62.capacity_table' does not depend on any axioms
'ChryseneFormalism.Article62.metric_channel_fibre' does not depend on any axioms
'ChryseneFormalism.Article62.full_engagement_halves' does not depend on any axioms
'ChryseneFormalism.Article62.no_engagement_retains_nothing' does not depend on any axioms
```

Exit code 0. Every theorem reports `does not depend on any axioms`: not `propext`, not `Classical.choice`, and not `sorryAx`.

How to read this log, and how to check it. The phrase *does not depend on any axioms* is Lean's strongest report. Its weaker forms are worth recognising: `propext` and `Classical.choice` are standard and harmless, while `sorryAx` means the theorem was *admitted rather than proved* — Lean emits an error for the failed tactic but still accepts the declaration, so a theorem can appear in a file, be cited in a table, and not be proved at all. A reader who wants to confirm the present dossier should run the file themselves and compare against the lines above; a reader who wants to confirm that this check is not vacuous should alter one of the defining constants, for instance `idxC12`, and observe the kernel refuse the result.