Executable Certification

Standalone Lean 4 Formal-Proof Dossier

Charles D. Schaper, Ph.D.
Chief Editor, Annals of the Chrysene Formalism
editor@chrysene.com

Companion to Article 101, "Executable Certification: Verification Code and Machine-Checked Proof as Accretive Layers in Orthogonal Review." The article occupies pages 1955–1965 in Volume 1, Number 5 (November 2026). This supplement has its own page numbering.

1 Scope and interpretation

The domains are an explicit editorial policy. The finite [A] category is assigned bounded exhaustive evaluation; the universal-discrete [A] category is assigned symbolic proof. Arithmetic [D] statements are checked relative to stated constants. The tags are supplied data, not an automatic classifier of arbitrary prose. The non-nesting result is conditional on this declared policy and its two witness classes. It is not a computability separation between programming languages or proof assistants.

The checks certify the formulas and finite objects represented below. They do not establish that a manuscript's extraction or classification is faithful. They do not certify molecular realization, chemical feasibility, statistical independence, or empirical truth.

Historical claims about Articles 41, 62 and 64 remain source reports: their reported audit counts, authorship independence, past compilation failures, 108-check harness, 23-theorem artifact and mutation outcomes are not rerun here. The pool ratio near 66, the 143-second replication threshold, the membrane tilt window and depth/footprint values require source constants, models or evidence not contained in this editorial. Numerical comparisons below verify only the explicitly stated arithmetic, not those derivations. The perimeter model is the abstract cycle on 18 numbered vertices; these vertex IDs are not a molecular locant assignment or a full chrysene graph.

2 Formal coverage

Lean 4.24.0 checks 31 named theorems:

- seven results about the six-kind policy, including both directions of non-nesting and the exact finite-[A] overlap;
- four general predicate-lifting results: vacuity, elimination, rejection of a failing in-domain claim, and the union/conjunction law;
- four admissibility results retaining the three original review gates and both new gates, including rejection when chemistry fails;
- the capacity identity $4^m = 2^m 2^m$ for every natural number m;
- four exhaustive perimeter results, including all 18 single-edge cuts and all 153 two-edge cuts;
- eight exact integer or scaled-integer arithmetic witnesses;
- three general results for an asserting-program contract, linking zero exit status to its declared predicate and rejecting an invalid mutation of a valid baseline.

For the cycle results, each cut produces explicit vertex walks. The checker requires nonempty disjoint coverage of the 18 vertices and equality of the walk-edge set with the uncut cycle edges. Single-cut walks are Hamiltonian paths; two-cut walks form two path components, allowing a singleton path. The proof is exhaustive kernel evaluation, not a test of sampled cuts.

The capacity theorem uses the standard natural-number multiplication-power identity and its transitive audit includes `propext`. The other 30 theorems report no axioms. No theorem depends on `sorryAx`; there are no admitted declarations, custom axioms or native-evaluation shortcuts in the distributed source. The library import `Std` is bundled with Lean 4.24.0; no Mathlib or network package dependency is used.

3 Execution and mutation sensitivity

Run `lean FormalProof_A101.lean` with Lean 4.24.0. Every theorem has a `#print axioms` audit in the source. A zero exit status and the dependency audit are both required. The five formal mutations widen the verification domain, change the capacity base, alter the cycle size, halve the claimed period or change the tether extension. The theorem statements and proofs are retained. Each mutated source must exit nonzero and report at least one failed theorem through its audit. These intentionally invalid files are test fixtures, not certified proofs. Run `python3 run_lean_checks.py /path/to/lean`. The runner generates them and stores their full compiler output.

```
Lean (version 4.24.0, x86_64-unknown-linux-gnu, commit 797c613eb9b6d4ec95db23e3e00af9ac6657f24b, Release)
Baseline Python: PASS; baseline Lean: PASS; audited theorems: 31
Python pair_count: 1 failed checks/theorems; exit 1
Python half_period: 1 failed checks/theorems; exit 1
Python bearing_sign: 2 failed checks/theorems; exit 1
Python tether_extension: 1 failed checks/theorems; exit 1
Python coverage_numerator: 1 failed checks/theorems; exit 1
Python unqualified_A_domain: 2 failed checks/theorems; exit 1
Python capacity_base: 32 failed checks/theorems; exit 1
Python energy_square: 96 failed checks/theorems; exit 1
Lean unqualified_A_domain: 3 failed checks/theorems; exit 1
Lean capacity_base: 1 failed checks/theorems; exit 1
Lean cycle_vertices: 2 failed checks/theorems; exit 1
Lean half_period: 1 failed checks/theorems; exit 1
Lean tether_extension: 1 failed checks/theorems; exit 1
All 13 expected-invalid mutations rejected. Exit status: 0
```

4 Exact checked source

```
import Std

/- Article 101: scoped editorial model and exact combinatorial witnesses.
   Categories are declared policy tags, not a classifier of arbitrary prose. -/
namespace Article101

inductive Kind where
  | finiteA | universalA | arithmeticD | otherD | importedC | molecularE
  deriving DecidableEq, Repr

open Kind

def V (k : Kind) : Prop := k = finiteA ∨ k = arithmeticD
def L (k : Kind) : Prop := k = finiteA ∨ k = universalA
instance (k : Kind) : Decidable (V k) := inferInstanceAs (Decidable (_ ∨ _))
instance (k : Kind) : Decidable (L k) := inferInstanceAs (Decidable (_ ∨ _))

theorem finite_overlap : V finiteA ∧ L finiteA := by decide
theorem universal_only_L : L universalA ∧ ¬ V universalA := by decide
theorem arithmetic_only_V : V arithmeticD ∧ ¬ L arithmeticD := by decide
theorem excluded_kinds : ¬ V otherD ∧ ¬ L otherD ∧ ¬ V importedC ∧
    ¬ L importedC ∧ ¬ V molecularE ∧ ¬ L molecularE := by decide

theorem exact_overlap (k : Kind) : (V k ∧ L k) ↔ k = finiteA := by
  cases k <;> decide

theorem L_not_subset_V : ¬ (∀ k, L k → V k) := by
  intro h
  exact universal_only_L.2 (h universalA universal_only_L.1)

theorem V_not_subset_L : ¬ (∀ k, V k → L k) := by
  intro h
  exact arithmetic_only_V.2 (h arithmeticD arithmetic_only_V.1)

-- This general schema does not depend on the six-kind finite model.
def Lift {α : Type} (manuscript domain passed : α → Prop) : Prop :=
  ∀ s, manuscript s → domain s → passed s

theorem lift_vacuous {α : Type} (m d p : α → Prop)
    (empty : ∀ s, m s → ¬ d s) : Lift m d p := by
  intro s hm hd
  exact False.elim (empty s hm hd)

theorem lift_elim {α : Type} (m d p : α → Prop)
    (h : Lift m d p) (s : α) (hm : m s) (hd : d s) : p s := h s hm hd

theorem lift_failing_claim {α : Type} (m d p : α → Prop)
    (s : α) (hm : m s) (hd : d s) (hp : ¬ p s) : ¬ Lift m d p := by
  intro h
  exact hp (h s hm hd)

theorem lift_union {α : Type} (m d e p : α → Prop) :
    Lift m (fun s => d s ∨ e s) p ↔ Lift m d p ∧ Lift m e p := by
  constructor
  · intro h
    exact ⟨fun s hm hd => h s hm (Or.inl hd),
      fun s hm he => h s hm (Or.inr he)⟩
  · intro h s hm hd
    cases hd with
    | inl hd => exact h.1 s hm hd
    | inr he => exact h.2 s hm he

def Admissible (math concordance chemistry v l : Prop) : Prop :=
  math ∧ concordance ∧ chemistry ∧ v ∧ l

theorem retain_original (m c h v l : Prop) (a : Admissible m c h v l) :
    m ∧ c ∧ h := ⟨a.1, a.2.1, a.2.2.1⟩

theorem retain_V (m c h v l : Prop) (a : Admissible m c h v l) : v := a.2.2.2.1

theorem retain_L (m c h v l : Prop) (a : Admissible m c h v l) : l := a.2.2.2.2

theorem no_tradeoff (m c h v l : Prop) (nh : ¬ h) : ¬ Admissible m c h v l := by
  intro a
  exact nh a.2.2.1

-- An all-natural theorem, not a bounded enumeration.
def capacityBase : Nat := 4

```

```
theorem capacity_dichotomy (m : Nat) : capacityBase ^ m = 2 ^ m * 2 ^ m := by
  change (2 * 2) ^ m = 2 ^ m * 2 ^ m
  exact Nat.mul_pow 2 2 m

-- Pure perimeter graph: vertex IDs 0,...,17, not chemical locant labels.
def n : Nat := 18
def vertices : List Nat := List.range n
def edge (a b : Nat) : Nat × Nat := (min a b, max a b)
def cycleEdge (i : Nat) : Nat × Nat := edge i ((i + 1) % n)
def walk (start len : Nat) : List Nat :=
  (List.range len).map (fun k => (start + k) % n)
def pathEdges : List Nat → List (Nat × Nat)
  | a :: b :: rest => edge a b :: pathEdges (b :: rest)
  | _ => []
def remaining (cuts : List Nat) : List (Nat × Nat) :=
  (vertices.filter (fun i => !cuts.contains i)).map cycleEdge

def sameEdges (a b : List (Nat × Nat)) : Bool :=
  a.length == b.length && a.all (fun e => b.contains e) &&
  b.all (fun e => a.contains e)

def singleOK (i : Nat) : Bool := Id.run do
  let p := walk (i + 1) n
  return p.length == n && decide p.Nodup &&
    vertices.all (fun v => p.contains v) &&
    sameEdges (pathEdges p) (remaining [i])

def pairs : List (Nat × Nat) := vertices.flatMap fun i =>
  (vertices.filter (fun j => i < j)).map (fun j => (i,j))

def pairOK (ij : Nat × Nat) : Bool := Id.run do
  let (i,j) := ij
  let p := walk (i + 1) (j - i)
  let q := walk (j + 1) (n - (j - i))
  return p.length > 0 && q.length > 0 &&
    (p ++ q).length == n && decide (p ++ q).Nodup &&
    vertices.all (fun v => (p ++ q).contains v) &&
    sameEdges (pathEdges p ++ pathEdges q) (remaining [i,j])

set_option maxRecDepth 100000
set_option maxHeartbeats 0

theorem eighteen_vertices : vertices.length = 18 := by decide
theorem eighteen_single_cuts : vertices.all singleOK = true := by decide
theorem one_hundred_fifty_three_pairs : pairs.length = 153 := by decide
theorem exhaustive_two_cut_decomposition : pairs.all pairOK = true := by decide

-- Exact rational witnesses are encoded by integer-scaled equalities.
def activeHours : Nat := 12
def secondsPerHour : Nat := 3600
def periodSeconds : Nat := 86400
def bearingTenths : Int := 1339
def contourTenths : Int := 114
def extensionTenths : Int := 38

theorem active_phase_seconds : activeHours * secondsPerHour = 43200 := by decide
theorem half_duty_period : 2 * (activeHours * secondsPerHour) = periodSeconds := by decide
theorem supplementary_angle : 1800 - bearingTenths = 461 := by decide
theorem perpendicular_bearing : bearingTenths - 900 = 439 := by decide
theorem window_span : (90 : Nat) - 46 = 44 := by decide
theorem tether_contraction : contourTenths - extensionTenths = 76 := by decide
-- 55/63, expressed as a percentage, lies in the rounding interval for 87.3%.
theorem coverage_rounding : (8725 : Nat) * 63 < 550000 ∧
    550000 < 8735 * 63 := by decide
theorem explicit_fraction_reduction : (48 : Nat) * 21 = 16 * 63 := by decide

-- Contract for a terminating assertion program relative to its declared Q.
def AssertingFor {Data : Type} (exit : Data → Nat) (Q : Data → Prop) : Prop :=
  ∀ d, (exit d = 0 ↔ Q d)

theorem accepted_iff_claim {Data : Type} (exit : Data → Nat) (Q : Data → Prop)
    (contract : AssertingFor exit Q) (d : Data) : exit d = 0 ↔ Q d := contract d

theorem invalid_mutation_rejected {Data : Type} (exit : Data → Nat) (Q : Data → Prop)
    (contract : AssertingFor exit Q) (d : Data) (mu : Data → Data)
    (invalid : ¬ Q (mu d)) : exit (mu d) ≠ 0 := by
  intro accepted
  exact invalid ((contract (mu d)).1 accepted)

theorem baseline_and_mutation {Data : Type} (exit : Data → Nat) (Q : Data → Prop)
    (contract : AssertingFor exit Q) (d : Data) (mu : Data → Data)
    (valid : Q d) (invalid : ¬ Q (mu d)) : exit d = 0 ∧ exit (mu d) ≠ 0 :=
```

```
  ⟨(contract d).2 valid, invalid_mutation_rejected exit Q contract d mu invalid⟩

end Article101

-- Verbatim transitive dependency audits for every named theorem.
#print axioms Article101.finite_overlap
#print axioms Article101.universal_only_L
#print axioms Article101.arithmetic_only_V
#print axioms Article101.excluded_kinds
#print axioms Article101.exact_overlap
#print axioms Article101.L_not_subset_V
#print axioms Article101.V_not_subset_L
#print axioms Article101.lift_vacuous
#print axioms Article101.lift_elim
#print axioms Article101.lift_failing_claim
#print axioms Article101.lift_union
#print axioms Article101.retain_original
#print axioms Article101.retain_V
#print axioms Article101.retain_L
#print axioms Article101.no_tradeoff
#print axioms Article101.capacity_dichotomy
#print axioms Article101.eighteen_vertices
#print axioms Article101.eighteen_single_cuts
#print axioms Article101.one_hundred_fifty_three_pairs
#print axioms Article101.exhaustive_two_cut_decomposition
#print axioms Article101.active_phase_seconds
#print axioms Article101.half_duty_period
#print axioms Article101.supplementary_angle
#print axioms Article101.perpendicular_bearing
#print axioms Article101.window_span
#print axioms Article101.tether_contraction
#print axioms Article101.coverage_rounding
#print axioms Article101.explicit_fraction_reduction
#print axioms Article101.accepted_iff_claim
#print axioms Article101.invalid_mutation_rejected
#print axioms Article101.baseline_and_mutation
```

5 Verbatim baseline output

Toolchain version:

```
Lean (version 4.24.0, x86_64-unknown-linux-gnu, commit 797c613eb9b6d4ec95db23e3e00af9ac6657f24b, Release)
```

```
'Article101.finite_overlap' does not depend on any axioms
'Article101.universal_only_L' does not depend on any axioms
'Article101.arithmetic_only_V' does not depend on any axioms
'Article101.excluded_kinds' does not depend on any axioms
'Article101.exact_overlap' does not depend on any axioms
'Article101.L_not_subset_V' does not depend on any axioms
'Article101.V_not_subset_L' does not depend on any axioms
'Article101.lift_vacuous' does not depend on any axioms
'Article101.lift_elim' does not depend on any axioms
'Article101.lift_failing_claim' does not depend on any axioms
'Article101.lift_union' does not depend on any axioms
'Article101.retain_original' does not depend on any axioms
'Article101.retain_V' does not depend on any axioms
'Article101.retain_L' does not depend on any axioms
'Article101.no_tradeoff' does not depend on any axioms
'Article101.capacity_dichotomy' depends on axioms: [propext]
'Article101.eighteen_vertices' does not depend on any axioms
'Article101.eighteen_single_cuts' does not depend on any axioms
'Article101.one_hundred_fifty_three_pairs' does not depend on any axioms
'Article101.exhaustive_two_cut_decomposition' does not depend on any axioms
'Article101.active_phase_seconds' does not depend on any axioms
'Article101.half_duty_period' does not depend on any axioms
'Article101.supplementary_angle' does not depend on any axioms
'Article101.perpendicular_bearing' does not depend on any axioms
'Article101.window_span' does not depend on any axioms
'Article101.tether_contraction' does not depend on any axioms
'Article101.coverage_rounding' does not depend on any axioms
'Article101.explicit_fraction_reduction' does not depend on any axioms
'Article101.accepted_iff_claim' does not depend on any axioms
'Article101.invalid_mutation_rejected' does not depend on any axioms
'Article101.baseline_and_mutation' does not depend on any axioms
```

Baseline process exit status: 0.

6 Reproducibility and byte identity

The source block above is inserted programmatically from the same UTF-8 file that produced the log. Extraction of the block is compared byte-for-byte with that file during the dossier build. The `extract_dossiers.py` utility reproduces that check. Compile this document directly with pdfLaTeX; no local font map or companion source file is needed for its LaTeX build. The executable source and logs are also distributed as plain files for convenience.

Source SHA-256:

```
f3cf9a5b7ff691325f928627d95e42bf58375f13c96d3b041fa66eb6972c24c7
```