Formal Proof Supplement

Symmetry-Compatible Information in the Chrysene Formalism

Axiomatic Realization, Coding Obstructions, Constructive Repair, and Falsifiable Molecular Models

Charles D. Schaper, Ph.D.

Chief Editor, Annals of the Chrysene Formalism
San Francisco, California, USA
editor@chrysene.com

Scope. This supplement presents a Lean 4 development of the concrete finite construction and two conditional decoding lemmas. Finite certificates use kernel reduction, not native evaluation as a proof oracle. The mathematical and empirical limits below are part of the certificate specification.

1 Toolchain and method

The pinned toolchain is Lean 4.24.0 (`leanprover/lean4:v4.24.0`); the only import is `Std`. No Mathlib installation is needed. Run `lean FormalProof_A100.lean`. Compilation may take several minutes because finite computations are checked by the kernel. There are no admitted proofs or project-defined axioms. The source concludes with `#print axioms` for every named theorem. The successful audit lists only the standard Lean axioms `propext`, `Classical.choice`, and `Quot.sound`; no admitted, custom, or native-evaluation axioms occur.

Field symbols are `Fin 4`; words are five-field records. The code is computed as the zero-syndrome filter of all words. Thus cardinality, covariance, decoder, and distance declarations are certificates about the actual defined operations, not propositions about stored answer arrays. Enumeration certificates are Boolean equalities over explicit finite lists. The manuscript identifies these lists with the intended finite spaces; this supplement does not provide a general-purpose finite-field or quotient-space library.

2 Declaration coverage

The development covers the field operation laws and Frobenius, reversal involution, 1,024 ambient words, both 64-word codes, the systematic encoder, reversal covariance and closure, 16 fixed codewords, 40 message orbits, 544 ambient orbits, four words in the original reversal intersection, and the explicit original distance-one obstruction. It also checks the repaired code's failure of conjugate reversal, all 16 error syndromes, ordinary and quotient distance bounds, decoder validity and equivariance on all received words, and projective matrix counts and parities. The final source contains 46 named theorem declarations, each followed by a dependency-audit command.

The general theorem `unique_radius_decode` proves uniqueness from a natural-number triangle inequality, symmetry, and distance separation greater than twice the radius. The theorem `equivariant_decoder` proves equivariance from isometry, code preservation, validity, and uniqueness. These are conditional theorems, not declarations that all hypotheses hold in every molecular model.

3 Explicit exclusions

The full abstract quotient-decoding theorem, projective classification and its reduction from arbitrary admissible parity-check matrices, and the universal semilinear no-go conclusion are not fully formalized here. Their finite instances or computational premises are checked. Shell counts and exhaustive equivalence of the radius-one union with the actual decoder-success predicate are checked in Python, not in Lean. Real-valued probability identities, information-theoretic entropy formulas, analytic feasibility bounds, chemical interpretation, and empirical validation remain outside this Lean development. No theorem asserts that substituted chrysene exists in a required device regime or caused biogenesis.

Schaper, C. D. (2026). Article 100 companion. CC BY-NC-ND 4.0.

4 Complete Lean source

The listing is generated verbatim from the executable source. Unicode in the source is retained; the plain `.lean` file is the authoritative executable.

```
import Std
set_option maxRecDepth 100000
set_option maxHeartbeats 0
namespace Article100
abbrev F := Fin 4
def add (a b : F) : F := ⟨Nat.xor a.val b.val % 4, Nat.mod_lt _ (by decide)⟩
def mul (a b : F) : F :=
  if a == 0 || b == 0 then 0 else
  if a == 1 then b else if b == 1 then a else
  if a == 2 then (if b == 2 then 3 else 1) else (if b == 2 then 1 else 2)
def sq (a : F) : F := mul a a
def fs : List F := [0,1,2,3]
structure W where
  a : F
  b : F
  c : F
  d : F
  e : F
  deriving DecidableEq, Repr
def words : List W := fs.flatMap fun a => fs.flatMap fun b => fs.flatMap fun c => fs.flatMap fun d => fs.map fun e => ⟨a,b,c,d,e⟩
def rev (x : W) : W := ⟨x.e,x.d,x.c,x.b,x.a⟩
def frob (x : W) : W := ⟨sq x.a,sq x.b,sq x.c,sq x.d,sq x.e⟩
def plus (x y : W) : W := ⟨add x.a y.a,add x.b y.b,add x.c y.c,add x.d y.d,add x.e y.e⟩
def ds (a b : F) : Nat := if a == b then 0 else 1
def dist (x y : W) : Nat := ds x.a y.a + ds x.b y.b + ds x.c y.c + ds x.d y.d + ds x.e y.e
def s1 (x : W) : F := add (add (add x.a x.b) x.d) x.e
def s2 (x : W) : F := add (add (add x.b x.c) (mul 2 x.d)) (mul 3 x.e)
def syn (x : W) : F × F := (s1 x,s2 x)
def old1 (x : W) : F := add (add (add x.a x.c) x.d) x.e
def oldSyn (x : W) : F × F := (old1 x,s2 x)
def code : List W := words.filter (fun x => syn x == (0,0))
def oldCode : List W := words.filter (fun x => oldSyn x == (0,0))
def enc (a b c : F) : W := ⟨add (add a (mul 3 b)) (mul 2 c),add (add a (mul 2 b)) (mul 3 c),a,b,c⟩
def idx (x : W) : Nat := ((((x.a.val*4+x.b.val)*4+x.c.val)*4+x.d.val)*4+x.e.val)
def canonical (x : W) : W := if idx x ≤ idx (rev x) then x else rev x
def qdist (x y : W) : Nat := min (dist x y) (dist x (rev y))
def oneErrors : List W := words.filter fun x => dist x ⟨0,0,0,0,0⟩ ≤ 1
def findError (s : F × F) : W := (oneErrors.find? fun e => syn e == s).getD ⟨0,0,0,0,0⟩
def decode (x : W) : W := plus x (findError (syn x))
def codeOrbits : List W := code.filter fun x => canonical x == x
def ambientOrbits : List W := words.filter fun x => canonical x == x

theorem add_comm : ∀ a b : F, add a b = add b a := by decide +kernel
theorem add_assoc : ∀ a b c : F, add (add a b) c = add a (add b c) := by decide +kernel
theorem add_zero : ∀ a : F, add a 0 = a := by decide +kernel
theorem add_self : ∀ a : F, add a a = 0 := by decide +kernel
theorem mul_comm : ∀ a b : F, mul a b = mul b a := by decide +kernel
theorem mul_assoc : ∀ a b c : F, mul (mul a b) c = mul a (mul b c) := by decide +kernel
theorem mul_one : ∀ a : F, mul a 1 = a := by decide +kernel
theorem mul_distrib : ∀ a b c : F, mul a (add b c) = add (mul a b) (mul a c) := by decide +kernel
theorem field_inverse : ∀ a : F, a ≠ 0 → ∃ b : F, mul a b = 1 := by decide +kernel
theorem frobenius_involution : ∀ a : F, sq (sq a) = a := by decide +kernel
theorem frobenius_additive : ∀ a b : F, sq (add a b) = add (sq a) (sq b) := by decide +kernel
theorem frobenius_multiplicative : ∀ a b : F, sq (mul a b) = mul (sq a) (sq b) := by decide +kernel
theorem rev_involution (x : W) : rev (rev x) = x := by cases x; rfl
theorem ambient_count : words.length = 1024 := by decide +kernel
theorem code_count : code.length = 64 := by decide +kernel
theorem original_count : oldCode.length = 64 := by decide +kernel
theorem encoder_valid : ∀ a b c : F, syn (enc a b c) = (0,0) := by decide +kernel
theorem encoder_systematic (a b c : F) : (enc a b c).c = a ∧ (enc a b c).d = b ∧ (enc a b c).e = c := by exact ⟨rfl,rfl,rfl⟩
theorem covariance_certificate : words.all (fun x => syn (rev x) == (s1 x, add (mul 3 (s1 x)) (s2 x))) = true := by decide +kernel
theorem reversal_closure : code.all (fun x => syn (rev x) == (0,0)) = true := by decide +kernel
theorem palindrome_count : (code.filter fun x => rev x == x).length = 16 := by decide +kernel
theorem message_orbit_count : codeOrbits.length = 40 := by decide +kernel
theorem ambient_orbit_count : ambientOrbits.length = 544 := by decide +kernel
theorem original_intersection_count : (oldCode.filter fun x => oldSyn (rev x) == (0,0)).length = 4 := by decide +kernel

def witnessC : W := ⟨0,0,1,2,3⟩
def witnessY : W := ⟨0,2,1,2,3⟩
theorem obstruction_membership : oldSyn witnessC = (0,0) ∧ oldSyn (rev witnessY) = (0,0) := by decide +kernel
theorem obstruction_distinct_orbits : canonical witnessC ≠ canonical witnessY := by decide +kernel
theorem obstruction_distance : qdist witnessC witnessY = 1 := by decide +kernel
theorem repair_not_semilinear : syn (rev (frob (enc 0 0 1))) ≠ (0,0) := by decide +kernel
theorem one_error_count : oneErrors.length = 16 := by decide +kernel
theorem syndrome_complete : (fs.flatMap fun a => fs.map fun b => (a,b)).all (fun s => syn (findError s) == s) = true := by decide +
    kernel
theorem one_error_syndromes_unique : oneErrors.all (fun e => oneErrors.all (fun f => !(syn e == syn f) || e == f)) = true := by
```

```
     decide +kernel

-- Boolean certificates below use the definitions above, not stored answer arrays.
theorem distance_certificate : code.all (fun c => code.all (fun d => c == d || decide (dist c d ≥ 3))) = true := by decide +kernel
theorem decoder_valid_certificate : words.all (fun x => syn (decode x) == (0,0) && decide (dist x (decode x) ≤ 1)) = true := by
     decide +kernel
theorem decoder_equivariance_certificate : words.all (fun x => decode (rev x) == rev (decode x)) = true := by decide +kernel
theorem quotient_distance_certificate : codeOrbits.all (fun c => codeOrbits.all (fun d => c == d || decide (qdist c d ≥ 3))) = true
      := by decide +kernel

def projective : List (F × F) := [(1,0),(0,1),(1,1),(1,2),(1,3)]
def projectiveIndex (p : F × F) : Nat :=
  if p.1 == 0 then 1 else if p.2 == 0 then 0 else
  if p.2 == p.1 then 2 else if p.2 == mul 2 p.1 then 3 else 4
structure Mat where
  a : F
  b : F
  c : F
  d : F
  deriving DecidableEq
def matrices : List Mat := fs.flatMap fun a => fs.flatMap fun b => fs.flatMap fun c => fs.map fun d => ⟨a,b,c,d⟩
def det (m : Mat) : F := add (mul m.a m.d) (mul m.b m.c)
def act (m : Mat) (p : F × F) : F × F := (add (mul m.a p.1) (mul m.b p.2),add (mul m.c p.1) (mul m.d p.2))
def perm (m : Mat) (semi : Bool) : List Nat := projective.map fun p => projectiveIndex (act m (if semi then (sq p.1,sq p.2) else p))
def inversions : List Nat → Nat
| [] => 0
| x::xs => (xs.filter fun y => y < x).length + inversions xs
def parity (p : List Nat) : Nat := inversions p % 2
def invertibles : List Mat := matrices.filter fun m => det m != 0
theorem invertible_count : invertibles.length = 180 := by decide +kernel
theorem all_linear_even : invertibles.all (fun m => parity (perm m false) == 0) = true := by decide +kernel
theorem all_semilinear_odd : invertibles.all (fun m => parity (perm m true) == 1) = true := by decide +kernel
theorem reversal_even : parity [4,3,2,1,0] = 0 := by decide +kernel
theorem projective_linear_count : (invertibles.map (fun m => perm m false)).eraseDups.length = 60 := by decide +kernel
theorem projective_semilinear_count : (invertibles.map (fun m => perm m true)).eraseDups.length = 60 := by decide +kernel

-- Generic uniqueness and equivariance consequences do not assume a particular field.
theorem unique_radius_decode {X : Type} (d : X → X → Nat) (C : X → Prop)
    (triangle : ∀ a b c, d a c ≤ d a b + d b c)
    (symm : ∀ a b, d a b = d b a) (t : Nat)
    (sep : ∀ a b, C a → C b → a ≠ b → 2*t < d a b)
    (a b y : X) (ha : C a) (hb : C b)
    (hya : d y a ≤ t) (hyb : d y b ≤ t) : a = b := by
  apply Classical.byContradiction
  intro h
  have h1 := sep a b ha hb h
  have h2 := triangle a y b
  have h3 := symm a y
  omega

theorem equivariant_decoder {X : Type} (d : X → X → Nat) (C : X → Prop)
    (r dec : X → X) (t : Nat)
    (preserves : ∀ x, C x → C (r x))
    (isometry : ∀ x y, d (r x) (r y) = d x y)
    (valid : ∀ y, C (dec y) ∧ d y (dec y) ≤ t)
    (unique : ∀ y a b, C a → C b → d y a ≤ t → d y b ≤ t → a = b)
    (y : X) : dec (r y) = r (dec y) := by
  apply unique (r y) _ _ (valid (r y)).1 (preserves _ (valid y).1) (valid (r y)).2
  rw [isometry]
  exact (valid y).2

theorem quotient_ball_arithmetic : 16*10 + 24*16 = (544 : Nat) := by decide +kernel
theorem word_prior_nonfixed : 64 - 16 = (48 : Nat) := by decide +kernel
theorem orbit_prior_nonfixed : 40 - 16 = (24 : Nat) := by decide +kernel

end Article100

-- Dependency audit for every named theorem.
#print axioms Article100.add_comm
#print axioms Article100.add_assoc
#print axioms Article100.add_zero
#print axioms Article100.add_self
#print axioms Article100.mul_comm
#print axioms Article100.mul_assoc
#print axioms Article100.mul_one
#print axioms Article100.mul_distrib
#print axioms Article100.field_inverse
#print axioms Article100.frobenius_involution
#print axioms Article100.frobenius_additive
#print axioms Article100.frobenius_multiplicative
#print axioms Article100.rev_involution
#print axioms Article100.ambient_count
#print axioms Article100.code_count
```

```
#print axioms Article100.original_count
#print axioms Article100.encoder_valid
#print axioms Article100.encoder_systematic
#print axioms Article100.covariance_certificate
#print axioms Article100.reversal_closure
#print axioms Article100.palindrome_count
#print axioms Article100.message_orbit_count
#print axioms Article100.ambient_orbit_count
#print axioms Article100.original_intersection_count
#print axioms Article100.obstruction_membership
#print axioms Article100.obstruction_distinct_orbits
#print axioms Article100.obstruction_distance
#print axioms Article100.repair_not_semilinear
#print axioms Article100.one_error_count
#print axioms Article100.syndrome_complete
#print axioms Article100.one_error_syndromes_unique
#print axioms Article100.distance_certificate
#print axioms Article100.decoder_valid_certificate
#print axioms Article100.decoder_equivariance_certificate
#print axioms Article100.quotient_distance_certificate
#print axioms Article100.invertible_count
#print axioms Article100.all_linear_even
#print axioms Article100.all_semilinear_odd
#print axioms Article100.reversal_even
#print axioms Article100.projective_linear_count
#print axioms Article100.projective_semilinear_count
#print axioms Article100.unique_radius_decode
#print axioms Article100.equivariant_decoder
#print axioms Article100.quotient_ball_arithmetic
#print axioms Article100.word_prior_nonfixed
#print axioms Article100.orbit_prior_nonfixed
```

Schaper, C. D. (2026). Article 100 companion. CC BY-NC-ND 4.0.