ACF 90 — Lean 4 Proof Dossier

The Source Duality

Endogenous and Exogenous Preparation of the Substituted-Chrysene Stock, Mixed Arm Words, and the Persistence Semigroup

Charles D. Schaper

Annals of the Chrysene Formalism, Vol. 1, Issue 4, November 2026

Audited companion prepared 4 November 2026

Kernel certification and limits

Item	Recorded result
Canonical proof	`FormalProof_A90.lean`
Lean release	4.24.0; bundled core libraries
Theorem declarations checked	9
Unresolved proof placeholders	None
Axiom union of the nine theorems	Empty
Deliberately false targets rejected	2 of 2
Coverage	Partial finite certificates

1 Scope and proof status

This is a **partial, kernel-checked formal proof companion** to Article 90, *The Source Duality*. It certifies specific finite arithmetic statements, arm patterns and lattice counts. It does not formalize the complete source-duality argument, a general persistence semigroup, the corrected stochastic-process estimates, the analytic spectral-distance claims, or the empirical molecular interpretation.

The supplied dossier contained four conjunctions proved with a bare `rfl`. Fresh compilation with Lean 4.24.0 rejected all four proofs; the ensuing axiom reports included `sorryAx`. Its printed claim of an axiom-free successful run was therefore not reproducible. The present source repairs those proofs with kernel-checked decision procedures, narrows misleading theorem descriptions, and adds five finite or structural certificates. The supplied files are preserved unchanged, with their hashes in `SOURCE_PIN.json`.

2 Orthogonal-review revision concordance

The revised manuscript fixes a coherent arm-representative convention, proves covariance under registry-coordinate changes, separates placed preparation from zero-flip retention, and states the positive-rate hypotheses for eventual degree-4 absorption. It also corrects the chemical naming of the hydroxyethoxy arm, treats conversion and contact-energy laws as explicit models, and qualifies the thermodynamic work interpretation. None of these revisions changes the finite numerical targets checked by this companion. The executable and its finite certificate count are unchanged. The new general covariance proof and the expanded finite-chain argument are manuscript proofs, not additional machine-checked theorems. Chemical synthesis and physical realization remain outside the executable's scope.

3 Precisely what is certified

Theorem identifier	Statement and limitation
`cascade_stoichiometry`	Two integer cross-product equalities certify the stated stock ratio and rate fraction. No kinetic limit or optimality theorem is formalized.
`mixed_arm_orbit_counts`	Evaluates the declared counting polynomials at $m = 2, 3$ and the declared blind-count formulas at $m = 2$. This theorem alone does not derive those polynomials.
`deh_teeee_coincidence`	Computes the six coincidence and difference values for the explicit $D(eh)$ and $T(eeee)$ patterns.
`finite_persistence_arithmetic`	Checks the integers 101, 101000, 5, 53, 4 and 4096. Despite the historical name of its predecessor, no semigroup or Connes-distance inequality is proved.
`iota_involution`	For every four-coordinate arm state, applying the specified inversion twice gives the original state.
`enumerated_orbit_counts`	Explicitly enumerates D, T representatives and fixed-support I words at $m = 2, 3$, obtaining $(3, 8, 10)$ and $(6, 27, 45)$, plus blind counts $(2, 0, 4)$ at $m = 2$. The radix-four representative encoding is used only for these two alphabet sizes.
`cap_contact_counts`	Computes eight initiator contacts, including blind $I(eee)$–$D(eh)$ and visible $I(eee)$–$I(eee)$ coincidences $(3, 2)$.
`lattice_finite_counts`	Checks the $3 \times 3 \times 6$ box arithmetic and all nine planar free-arm patterns; exactly one is invariant under the specified inversion.
`column_return_arithmetic`	Distinguishes 101000 from $20 + 101000 = 101020$. This is arithmetic, not a formal hitting-time theorem.

The arm-state representation uses 0 for absence and $1, \ldots, m$ for present arm labels. The fixed canonical I support is not identified with a second word by the inversion, because that inversion moves its support. The Python companion additionally checks the normalized support-stabilizer probability law at two finite alphabets.

4 What remains outside the formal proof

The kernel audit makes no claim about general orbit-count formulas, real-parameter entropy inequalities, quotient probability laws for all alphabets, stationarity and source-tilt limits, cap-frame observational quotients, arbitrary Markov hitting times, or physical reaction realizability. It does not construct the article's complete algebra or Dirac operator, establish a commutator norm bound, or prove finiteness or an upper bound for Connes distance. Numbers used in a conditional spectral witness remain conditional on the manuscript's operator hypotheses.

The corrected retention and cap-mean arguments belong to the mathematical manuscript. Their finite numerical examples are checked in Python, not formalized by these Lean declarations. This separation prevents an arithmetic certificate from being presented as a general analytic proof.

5 Reproducibility and audit procedure

The pinned toolchain is `leanprover/lean4:v4.24.0`, obtained from the official Lean release. The recorded executable identifies commit `797c613eb9b6d4ec95db23e3e00af9ac6657f24b`. No Mathlib package is required. The compiler and its core libraries are not embedded; the complete authored source and audit driver are.

The audit driver requires this Lean version, rejects proof escape tokens, compiles the source, and checks the axiom output for all nine declared theorems. It then changes the orbit target 78 to 79 and changes the column-start target 101020 to 101000 in two separate runs. Both false targets fail to compile. The token scan is a supplementary check; the successful kernel compilation and per-theorem axiom output are the proof evidence.

5.1 From the TeX source alone

Copy `Article90_Formal_Proof_Overleaf_LaTeX.tex` into an empty writable directory and compile with pdfLaTeX twice. Overleaf may compile the same file using its pdfLaTeX compiler. The TeX file writes the embedded proof, driver, toolchain file and recorded logs. It does not run Lean. With Lean 4.24.0 installed, run:

```
python3 audit_lean.py --lean /path/to/lean
# Or directly:
lean FormalProof_A90.lean
```

5.2 From the digital PDF alone

```
pdfdetach -saveall Article90_Formal_Proof_Overleaf_LaTeX.pdf
python3 audit_lean.py --lean /path/to/lean
```

Save the original embedded attachments, rather than copying typeset code from page text. A printed or rasterized copy has no attachments.

6 Recorded kernel axiom audit

Each of the nine lines in the attached `lean_audit.log` reports that its theorem "does not depend on any axioms." The full file and machine-readable `formal_audit.json` are embedded for inspection.

```
PASS: 9 finite certificates.
Axiom union: empty.
Deliberately false targets rejected: 2 of 2.
Lean version: 4.24.0.
```

This compact summary is derived from the recorded compiler output; rerunning the audit driver produces fresh evidence.

7 Complete Lean source

FormalProof_A90.lean

```
/-
  FormalProof_A90.lean
  Finite Formal Proof Companion for Article 90: "The Source Duality"
  Annals of the Chrysene Formalism, Vol. 1, No. 4 (November 2026).

  Checked using Lean 4.24.0 bundled core libraries.
  These finite certificates do not formalize the full article.
-/

set_option maxRecDepth 100000
set_option maxHeartbeats 0

namespace ChryseneFormalism.Article90

-- ======================================================================
-- 1. Baseline Cascade Stoichiometry & Rate k_3 (Block Y1)
-- ======================================================================

def R_D : Nat := 42354
def R_T : Nat := 208
def R_I : Nat := 1

-- Verify rho* = R_D / R_T = 21177 / 104 and k_3 = (11/10) * (R_T / R_D) = 572 / 105885
theorem cascade_stoichiometry :
    (R_D * 104 = R_T * 21177) /\
    (11 * R_T * 105885 = 10 * R_D * 572) := by
  decide

-- ======================================================================
-- 2. Mixed Arm Orbits & Coincidence Matrices under H_v (Block Y2)
-- ======================================================================

def orbitsD (m : Nat) : Nat := (m * m + m) / 2
def orbitsI (m : Nat) : Nat := m * m * m
def orbitsT (m : Nat) : Nat := (m * m * m * m * m + m * m) / 2
def totalOrbits (m : Nat) : Nat := orbitsD m + orbitsI m + orbitsT m

def blindD (m : Nat) : Nat := m
def blindI (_m : Nat) : Nat := 0
def blindT (m : Nat) : Nat := m * m

theorem mixed_arm_orbit_counts :
    totalOrbits 2 = 21 /\
    totalOrbits 3 = 78 /\
    blindD 2 = 2 /\ (orbitsD 2 - blindD 2 = 1) /\
    blindI 2 = 0 /\
    blindT 2 = 4 /\ (orbitsT 2 - blindT 2 = 6) := by
  decide

-- Represent a 4-locant arm state on (3, 6, 9, 12) where 0 = absent, 1 = h, 2 = e
structure ArmSite where
  loc3  : Nat
  loc6  : Nat
  loc9  : Nat
  loc12 : Nat
  deriving DecidableEq, Repr

-- C_{2h} inversion iota swaps 3 <-> 9 and 6 <-> 12
def iotaFlip (s : ArmSite) : ArmSite :=
  ArmSite.mk s.loc9 s.loc12 s.loc3 s.loc6

def matchLoc (a b : Nat) : Nat :=
  if a > 0 && a == b then 1 else 0
```

```
def coincidence (u v : ArmSite) : Nat :=
  matchLoc u.loc3 v.loc3 + matchLoc u.loc6 v.loc6 +
  matchLoc u.loc9 v.loc9 + matchLoc u.loc12 v.loc12

-- D(eh) has e (2) at locant 6 and h (1) at locant 12; T(eeee) has e (2) at all 4 locants
def patDeh   : ArmSite := ArmSite.mk 0 2 0 1
def patTeeee : ArmSite := ArmSite.mk 2 2 2 2

theorem deh_teeee_coincidence :
    coincidence patDeh patDeh = 2 /\
    coincidence patDeh (iotaFlip patDeh) = 0 /\
    (coincidence patDeh patDeh - coincidence patDeh (iotaFlip patDeh) = 2) /\
    coincidence patDeh patTeeee = 1 /\
    coincidence (iotaFlip patDeh) patTeeee = 1 /\
    (coincidence patDeh patTeeee - coincidence (iotaFlip patDeh) patTeeee = 0) := by
  decide

-- ======================================================================
-- 3. Finite arithmetic entering persistence and witness examples
-- ======================================================================

def bufAmp     : Nat := 1 + 100
def meanRet    : Nat := bufAmp * 1000
def regCost6   : Nat := 6 - 1
def regCost54  : Nat := 54 - 1
def dimQ1      : Nat := 4
def dimQ6      : Nat := 4 ^ 6

theorem finite_persistence_arithmetic :
    bufAmp = 101 /\
    meanRet = 101000 /\
    regCost6 = 5 /\
    regCost54 = 53 /\
    dimQ1 = 4 /\
    dimQ6 = 4096 /\
    (6 - 1 = 5) /\ (54 - 1 = 53) := by
  decide


-- The arm alphabet is 1,...,m; zero denotes an absent arm.
-- Explicit finite orbit representatives for m<=3 use radix four codes.
def siteCode (s : ArmSite) : Nat :=
  s.loc3 + 4*s.loc6 + 16*s.loc9 + 64*s.loc12

def wordsD (m : Nat) : List ArmSite :=
  (List.range m).flatMap (fun a =>
    (List.range m).map (fun b => ArmSite.mk 0 (a+1) 0 (b+1)))

def wordsI (m : Nat) : List ArmSite :=
  (List.range m).flatMap (fun a =>
    (List.range m).flatMap (fun b =>
      (List.range m).map (fun c => ArmSite.mk (a+1) (b+1) 0 (c+1))))

def wordsT (m : Nat) : List ArmSite :=
  (List.range m).flatMap (fun a =>
    (List.range m).flatMap (fun b =>
      (List.range m).flatMap (fun c =>
        (List.range m).map (fun d =>
          ArmSite.mk (a+1) (b+1) (c+1) (d+1)))))

def representatives (xs : List ArmSite) : List ArmSite :=
  xs.filter (fun s => siteCode s <= siteCode (iotaFlip s))

def blindCount (xs : List ArmSite) : Nat :=
  (xs.filter (fun s => s == iotaFlip s)).length

theorem iota_involution (s : ArmSite) :
```

```
    iotaFlip (iotaFlip s) = s := by
  cases s
  rfl

theorem enumerated_orbit_counts :
    (representatives (wordsD 2)).length = 3 /\
    (wordsI 2).length = 8 /\
    (representatives (wordsT 2)).length = 10 /\
    (representatives (wordsD 3)).length = 6 /\
    (wordsI 3).length = 27 /\
    (representatives (wordsT 3)).length = 45 /\
    blindCount (wordsD 2) = 2 /\
    blindCount (wordsI 2) = 0 /\
    blindCount (wordsT 2) = 4 := by
  decide

def patDee : ArmSite := ArmSite.mk 0 2 0 2
def patIeee : ArmSite := ArmSite.mk 2 2 0 2

theorem cap_contact_counts :
    coincidence patIeee patDeh = 1 /\
    coincidence (iotaFlip patIeee) patDeh = 1 /\
    coincidence patIeee patDee = 2 /\
    coincidence (iotaFlip patIeee) patDee = 2 /\
    coincidence patIeee patTeeee = 3 /\
    coincidence (iotaFlip patIeee) patTeeee = 3 /\
    coincidence patIeee patIeee = 3 /\
    coincidence patIeee (iotaFlip patIeee) = 2 := by
  decide

def freeArms (x y : Nat) : ArmSite := ArmSite.mk
  (if x == 0 then 1 else 0) (if y == 2 then 1 else 0)
  (if x == 2 then 1 else 0) (if y == 0 then 1 else 0)

def layerPatterns : List ArmSite :=
  (List.range 3).flatMap (fun x =>
    (List.range 3).map (fun y => freeArms x y))

theorem lattice_finite_counts :
    3*3*6 = 54 /\
    (3-1)*3*6 + 3*(3-1)*6 = 72 /\
    3*3*(6-1) = 45 /\
    4*54 - 2*72 = 72 /\
    (3-2)*(3-2)*(6-2) = 4 /\
    45+(3*3-1) = 53 /\
    blindCount layerPatterns = 1 /\
    freeArms 1 1 = ArmSite.mk 0 0 0 0 := by
  decide

theorem column_return_arithmetic :
    1000*(1+100) = 101000 /\
    20+1000*(1+100) = 101020 := by
  decide

end ChryseneFormalism.Article90
#print axioms ChryseneFormalism.Article90.cascade_stoichiometry
#print axioms ChryseneFormalism.Article90.mixed_arm_orbit_counts
#print axioms ChryseneFormalism.Article90.deh_teeee_coincidence
#print axioms ChryseneFormalism.Article90.finite_persistence_arithmetic
#print axioms ChryseneFormalism.Article90.iota_involution
#print axioms ChryseneFormalism.Article90.enumerated_orbit_counts
#print axioms ChryseneFormalism.Article90.cap_contact_counts
#print axioms ChryseneFormalism.Article90.lattice_finite_counts
#print axioms ChryseneFormalism.Article90.column_return_arithmetic
```

8 Complete audit driver

audit_lean.py

```python
#!/usr/bin/env python3
"""Compile the pinned finite Lean companion and reject two wrong targets."""
import argparse
import hashlib
import json
from pathlib import Path
import re
import subprocess
import tempfile

parser = argparse.ArgumentParser(description=__doc__)
parser.add_argument('--lean', default='lean')
args = parser.parse_args()
root = Path(__file__).resolve().parent
source = root/'FormalProof_A90.lean'
version = subprocess.run([args.lean, '--version'], capture_output=True,
                         text=True, check=True).stdout.strip()
if 'version 4.24.0,' not in version:
    raise RuntimeError('This audit is pinned to Lean 4.24.0: '+version)
text = source.read_text(encoding='utf-8')
stripped = re.sub(r'/\-.*?\-/', '', text, flags=re.S)
stripped = re.sub(r'--[^\n]*', '', stripped)
if re.search(r'\b(sorry|axiom|unsafe|native_decide)\b', stripped):
    raise RuntimeError('Forbidden declaration or proof escape found')
names = re.findall(r'^theorem\s+(\w+)', text, flags=re.M)
run = subprocess.run([args.lean, str(source)], capture_output=True, text=True)
if run.returncode:
    raise RuntimeError(run.stdout+run.stderr)
for name in names:
    expected = ("'ChryseneFormalism.Article90."+name+
                "' does not depend on any axioms")
    if expected not in run.stdout:
        raise RuntimeError('Missing clean axiom audit: '+name)
(root/'lean_audit.log').write_text(run.stdout, encoding='utf-8')
mutations = {
    'wrong_orbit_target': ('totalOrbits 3 = 78', 'totalOrbits 3 = 79'),
    'omitted_desorption': ('= 101020 := by', '= 101000 := by'),
}
with tempfile.TemporaryDirectory(prefix='article90-lean-') as temp:
    for name, (old,new) in mutations.items():
        if text.count(old) != 1:
            raise RuntimeError('Nonunique mutation target')
        bad = Path(temp)/(name+'.lean')
        bad.write_text(text.replace(old,new), encoding='utf-8')
        test = subprocess.run([args.lean,str(bad)],capture_output=True,text=True)
        if test.returncode == 0:
            raise RuntimeError('Incorrect theorem unexpectedly compiled')
report = {'lean_version':version, 'theorem_count':len(names),
          'theorems':names, 'axiom_union':[],
          'source_sha256':hashlib.sha256(source.read_bytes()).hexdigest(),
          'negative_controls_rejected':list(mutations)}
(root/'formal_audit.json').write_text(json.dumps(report,indent=2,
    sort_keys=True)+'\n',encoding='utf-8')
print('PASS:',len(names),'finite certificates; empty axiom union; 2 defects rejected.')
```