Quantum Behavior of Substituted-Chrysene Molecular Assemblies

Bipartite Transfer, Stark Response, and Registry-Gated Transport

Lean 4 Formal Proof Companion

Charles D. Schaper, Ph.D.
Annals of the Chrysene Formalism, Volume 1, Number 4
Article 93, November 2026

1 Formal scope and result

Lean 4.24.0 freshly compiled **57 named finite theorems**. Every theorem's `#print axioms` audit reports **no axiom dependencies**. The source imports only `Std` and uses kernel reduction through `decide`; it contains no `sorry`, `admit`, custom axioms or `native_decide`.

This is a **partial formal proof companion**. It certifies the explicit graph, integer matrix identities, matching counts, finite path recurrences, inversion geometry and finite transmission arithmetic. The rational polarizability enclosure is checked by Python and explained in that companion. Analytic resolvent identities, continuum scattering derivations, response theory, all-length asymptotics and chemical realization are not claimed as Lean-formalized here.

2 Concrete definitions and theorem coverage

Graph and exact inverse.
The full adjacency matrix and $C = 8A^{-1}$ are explicit integer matrices in the manuscript's vertex order. Lean proves symmetry, bipartition, 42 directed edge incidences, $AC = CA = 8I$, the same-sublattice zero blocks and the selected entries $C_y = 4, C_x = 1$. The interpretation as a rational inverse uses the nonzero scalar 8. The determinant -64 is checked by Python, not presented as a separate Lean determinant theorem.

Walk certificates.
Explicit row vectors for powers zero through five and seven are checked against the initial basis vectors and each adjacency recurrence. The shorter entries vanish and the first nonzero entries are 3 and 1. The general equivalence of adjacency powers and graph walks is standard mathematics used in the manuscript, not formalized as a generic graph library here.

Perfect matchings.
A terminating recursive enumeration selects the first remaining vertex, sums over adjacent partners, and recurses on the remaining vertices with decreasing fuel. For these graphs Lean evaluates eight complete matchings and residual counts four and one. Correctness of this enumeration as a general matching algorithm is explained by the first-pair partition, not proved as an abstract combinatorial theorem.

Inversion geometry.
Integer arrays encode four times the x and $y/\sqrt{3}$ coordinates. Lean checks the graph involution, coordinate sign reversal, D/T invariant supports and I/Dm noninvariant supports. It proves the scaled geometric-vector sums, lengths and inner product. These geometric certificates do not assert an electronic dipole or a molecular conformation.

Finite chain arithmetic.
Binary contact lists are enumerated explicitly. The six-site histogram is $(10, 15, 6, 1)$ and

the normalized mean is verified by cross multiplication. At $\tau = 1/2$, the transmission is represented as the ratio $4^{d+1}/(4^d + 1)^2$, $d = |\Delta m|$. Reversal and the upper bound are checked on all odd contact lengths 1,3,5,7,9. The SSH examples and an imperfect-transmission witness are also certified. These declarations are finite cases, not an all-length scattering theorem.

3 Reproduction and trust boundary

Run `lean FormalProof_A93.lean` with Lean 4.24.0, or run `python audit_lean.py` with `LEAN` set to that compiler. The audit checks the SHA-256 source pin, compiles the exact source, requires an axiom-free audit line for every named theorem, and records the compiler version, hash, theorem names and output. There are no external Lean libraries.

The standalone LaTeX source embeds the identical Lean source as Base64 and emits that file on compilation. Decode it to recover the source, or use the directly supplied `.lean` file. The package includes the toolchain declaration, source hash, runner and fresh compilation log. Overleaf typesetting does not compile Lean.

4 Concordance with the revised manuscript

The finite proofs retain the declared graph model and support the corrected transport counts. They distinguish bare transfer from finite-contact conductance, geometric vectors from dipoles, and perfect transmission from unitarity. They do not validate the inherited hardcoded polarizability values; those have been replaced by the computed values and exact rational enclosures in the verification companion. The manuscript's chemical assumptions remain explicit and separate from these certificates.

5 Complete Lean source

FormalProof_A93.lean

```
import Std
/- Article 93 finite exact certificates. Lean 4.24.0, standard library.
No floating spectra, real analysis, universal scattering, or chemical claims.
All computational proofs use kernel reduction (decide), not native_decide. -/
set_option maxRecDepth 100000
set_option maxHeartbeats 0
namespace Article93
abbrev Mat := List (List Int)
def get (m : Mat) (i j : Nat) : Int := (m[i]?.getD [])[j]?.getD 0
def ident (scale : Int) : Mat := (List.range 18).map fun i =>
 (List.range 18).map fun j => if i == j then scale else 0
def mul (a b : Mat) : Mat := (List.range 18).map fun i =>
 (List.range 18).map fun j => ((List.range 18).map fun k => get a i k * get b k j).sum
def adjacency : Mat := [
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0],
 [1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
def inverseTimesEight : Mat := [
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 5, -5, -2, 1, -1, -3, 3, 1, 2],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 5, 2, -1, 1, 3, -3, -1, -2],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, 6, -3, 3, 1, -1, -3, 2],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, -1, -2, 5, 3, 1, -1, -3, 2],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 2, -5, 5, -1, 1, 3, -2],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, -2, 2, 4, -2, 2, 6, 2, -2, -4],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 2, -2, -4, 2, -2, 2, -2, 2, 4],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, -3, 3, -2, 1, -1, -3, 3, 1, 2],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 1, 2, 3, -3, -1, 1, 3, -2],
 [5, 3, 1, 1, -1, -2, 2, -3, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [-5, 5, -1, -1, 1, 2, -2, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [-2, 2, 6, -2, 2, 4, -4, -2, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [1, -1, -3, 5, -5, -2, 2, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [-1, 1, 3, 3, 5, 2, -2, -1, -3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [-3, 3, 1, 1, -1, 6, 2, -3, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [3, -3, -1, -1, 1, 2, -2, 3, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [1, -1, -3, -3, 3, -2, 2, 1, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [2, -2, 2, 2, -2, -4, 4, 2, -2, 0, 0, 0, 0, 0, 0, 0, 0, 0]
]
theorem graph_symmetric : ((List.range 18).all fun i => (List.range 18).all fun j =>
```

```
 get adjacency i j == get adjacency j i) = true := by decide
theorem graph_bipartite : ((List.range 18).all fun i => (List.range 18).all fun j =>
 if (i<9) == (j<9) then get adjacency i j == 0 else true) = true := by decide
theorem edges_twice : (adjacency.map List.sum).sum = 42 := by decide
theorem inverse_left : mul adjacency inverseTimesEight = ident 8 := by decide
theorem inverse_right : mul inverseTimesEight adjacency = ident 8 := by decide
theorem inverse_same_sublattice : ((List.range 18).all fun i => (List.range 18).all fun j =>
 if (i<9) == (j<9) then get inverseTimesEight i j == 0 else true) = true := by decide
theorem green_y : get inverseTimesEight 5 11 = 4 := by decide
theorem green_x : get inverseTimesEight 1 13 = 1 := by decide
theorem cross_A : get inverseTimesEight 1 5 = 0 := by decide
theorem cross_B : get inverseTimesEight 13 11 = 0 := by decide
theorem amplitude_squared_ratio : 4*4 = 16*(1*1) := by decide
def step (v : List Int) : List Int := (List.range 18).map fun j =>
 ((List.range 18).map fun k => (v[k]?.getD 0) * get adjacency k j).sum
def initial (i : Nat) : List Int := (List.range 18).map fun j => if i == j then 1 else 0
def walks_y : Mat := [
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0],
 [1, 0, 0, 0, 0, 2, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 3, 4, 1, 2],
 [5, 2, 2, 0, 1, 7, 6, 7, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 9, 3, 1, 1, 13, 19, 8, 15]
]
theorem walks_y_initial : walks_y[0]?.getD [] = initial 5 := by decide
theorem walks_y_recurrence : ((List.range 5).all fun r => step (walks_y[r]?.getD []) ==
    walks_y[r+1]?.getD []) = true := by decide
theorem walks_y_no_shorter : ((List.range 5).all fun r => get walks_y r 11 == 0) = true :=
    by decide
theorem walks_y_first : get walks_y 5 11 = 3 := by decide
def walks_x : Mat := [
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0],
 [1, 2, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 3, 0, 0, 0, 0, 2, 0, 1],
 [5, 6, 1, 0, 0, 2, 1, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 11, 12, 1, 0, 0, 3, 13, 1, 8],
 [24, 23, 9, 0, 1, 16, 12, 33, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 47, 56, 11, 2, 1, 28, 73, 15, 54]
]
theorem walks_x_initial : walks_x[0]?.getD [] = initial 1 := by decide
theorem walks_x_recurrence : ((List.range 7).all fun r => step (walks_x[r]?.getD []) ==
    walks_x[r+1]?.getD []) = true := by decide
theorem walks_x_no_shorter : ((List.range 7).all fun r => get walks_x r 13 == 0) = true :=
    by decide
theorem walks_x_first : get walks_x 7 13 = 1 := by decide
def matchingCount : Nat → List Nat → Nat
 | 0, vs => if vs.isEmpty then 1 else 0
 | fuel+1, [] => 1
 | fuel+1, u::vs => (vs.map fun v => if get adjacency u v == 1 then
    matchingCount fuel (vs.filter fun w => w != v) else 0).sum
theorem perfect_matchings : matchingCount 9 (List.range 18) = 8 := by decide
theorem residual_y_matchings : matchingCount 8 ((List.range 18).filter fun v => v != 5 && v
    != 11) = 4 := by decide
theorem residual_x_matchings : matchingCount 8 ((List.range 18).filter fun v => v != 1 && v
    != 13) = 1 := by decide
def coordinateX4 : List Int := [-7, -1, 5, 5, -1, -7, -1, -1, 5, -5, 1, 7, 7, 1, -5, -5, 1,
    1]
```

```
def coordinateY4 : List Int := [5, 7, 1, -7, -5, 1, -1, 3, -3, 7, 5, -1, -5, -7, -1, 3, -3, 1]
def inversion : List Nat := [12, 13, 14, 9, 10, 11, 17, 16, 15, 3, 4, 5, 0, 1, 2, 8, 7, 6]
def invertIndex (i : Nat) := inversion[i]?.getD 0
theorem inversion_involutive : ((List.range 18).all fun i => invertIndex (invertIndex i) == i) = true := by decide
theorem inversion_graph : ((List.range 18).all fun i => (List.range 18).all fun j =>
 get adjacency (invertIndex i) (invertIndex j) == get adjacency i j) = true := by decide
theorem inversion_coordinates : ((List.range 18).all fun i =>
 coordinateX4[invertIndex i]?.getD 0 == -(coordinateX4[i]?.getD 0) &&
 coordinateY4[invertIndex i]?.getD 0 == -(coordinateY4[i]?.getD 0)) = true := by decide
def vectorSum (vs : List Nat) : Int × Int :=
 ((vs.map fun i => coordinateX4[i]?.getD 0).sum,
  (vs.map fun i => coordinateY4[i]?.getD 0).sum)
def sites_D : List Nat := [11, 5]
theorem vector_D : vectorSum sites_D = (0,0) := by decide
def sites_T : List Nat := [1, 11, 13, 5]
theorem vector_T : vectorSum sites_T = (0,0) := by decide
def sites_I : List Nat := [1, 11, 5]
theorem vector_I : vectorSum sites_I = (-1,7) := by decide
def sites_Dm : List Nat := [11]
theorem vector_Dm : vectorSum sites_Dm = (7,-1) := by decide
theorem dipole_I_norm_scaled : (-1 : Int)^2+3*7^2 = 148 := by decide
theorem dipole_Dm_norm_scaled : (7 : Int)^2+3*(-1)^2 = 52 := by decide
theorem dipole_dot_scaled : (-1 : Int)*7+3*7*(-1) = -28 := by decide
def invariant (vs : List Nat) : Bool := vs.all fun i => vs.contains (invertIndex i)
theorem D_invariant : invariant sites_D = true := by decide
theorem T_invariant : invariant sites_T = true := by decide
theorem I_not_invariant : invariant sites_I = false := by decide
theorem Dm_not_invariant : invariant sites_Dm = false := by decide

def words : Nat → List (List Nat)
 | 0 => [[]]
 | k+1 => (words k).flatMap fun t => [0::t,1::t]
def imbalance (xs : List Nat) : Int := ((xs.zipIdx.map fun p =>
 if p.2 % 2 == 0 then (p.1 : Int) else -(p.1 : Int))).sum
def countAbs (k d : Nat) : Nat := ((words k).filter fun xs => (imbalance xs).natAbs == d).length
-- T at tau=1/2 is represented by numerator/denominator (4^(d+1),(4^d+1)^2).
def transNum (d : Nat) := 4^(d+1)
def transDen (d : Nat) := (4^d+1)^2
theorem zero_transmission_balance : transNum 0 = transDen 0 := by decide
theorem SSH_four : transNum 2 * 289 = 64 * transDen 2 := by decide
theorem SSH_six : transNum 3 * 4225 = 256 * transDen 3 := by decide
theorem contacts_six : (words 5).length = 32 := by decide
theorem histogram_zero : countAbs 5 0 = 10 := by decide
theorem histogram_one : countAbs 5 1 = 15 := by decide
theorem histogram_two : countAbs 5 2 = 6 := by decide
theorem histogram_three : countAbs 5 3 = 1 := by decide
theorem histogram_total : 10+15+6+1 = 32 := by decide
theorem reversal_1_contacts : ((words 1).all fun xs => imbalance xs.reverse == imbalance xs) = true := by decide
theorem bound_1_contacts : ((words 1).all fun xs => transNum (imbalance xs).natAbs <= transDen (imbalance xs).natAbs) = true := by decide
theorem reversal_3_contacts : ((words 3).all fun xs => imbalance xs.reverse == imbalance xs) = true := by decide
theorem bound_3_contacts : ((words 3).all fun xs => transNum (imbalance xs).natAbs <= transDen (imbalance xs).natAbs) = true := by decide
```

```
theorem reversal_5_contacts : ((words 5).all fun xs => imbalance xs.reverse == imbalance xs)
     = true := by decide
theorem bound_5_contacts : ((words 5).all fun xs => transNum (imbalance xs).natAbs <=
    transDen (imbalance xs).natAbs) = true := by decide
theorem reversal_7_contacts : ((words 7).all fun xs => imbalance xs.reverse == imbalance xs)
     = true := by decide
theorem bound_7_contacts : ((words 7).all fun xs => transNum (imbalance xs).natAbs <=
    transDen (imbalance xs).natAbs) = true := by decide
theorem reversal_9_contacts : ((words 9).all fun xs => imbalance xs.reverse == imbalance xs)
     = true := by decide
theorem bound_9_contacts : ((words 9).all fun xs => transNum (imbalance xs).natAbs <=
    transDen (imbalance xs).natAbs) = true := by decide
theorem six_site_mean_crossmultiply :
 (10*25*289*4225 + 15*16*289*4225 + 6*64*25*4225 + 256*25*289) * 19536400 =
 12814237*(32*25*289*4225) := by decide
-- Finite obstruction to confusing unitarity with perfect transmission.
theorem imperfect_example : transNum 1 < transDen 1 := by decide
end Article93

#print axioms Article93.graph_symmetric
#print axioms Article93.graph_bipartite
#print axioms Article93.edges_twice
#print axioms Article93.inverse_left
#print axioms Article93.inverse_right
#print axioms Article93.inverse_same_sublattice
#print axioms Article93.green_y
#print axioms Article93.green_x
#print axioms Article93.cross_A
#print axioms Article93.cross_B
#print axioms Article93.amplitude_squared_ratio
#print axioms Article93.walks_y_initial
#print axioms Article93.walks_y_recurrence
#print axioms Article93.walks_y_no_shorter
#print axioms Article93.walks_y_first
#print axioms Article93.walks_x_initial
#print axioms Article93.walks_x_recurrence
#print axioms Article93.walks_x_no_shorter
#print axioms Article93.walks_x_first
#print axioms Article93.perfect_matchings
#print axioms Article93.residual_y_matchings
#print axioms Article93.residual_x_matchings
#print axioms Article93.inversion_involutive
#print axioms Article93.inversion_graph
#print axioms Article93.inversion_coordinates
#print axioms Article93.vector_D
#print axioms Article93.vector_T
#print axioms Article93.vector_I
#print axioms Article93.vector_Dm
#print axioms Article93.dipole_I_norm_scaled
#print axioms Article93.dipole_Dm_norm_scaled
#print axioms Article93.dipole_dot_scaled
#print axioms Article93.D_invariant
#print axioms Article93.T_invariant
#print axioms Article93.I_not_invariant
#print axioms Article93.Dm_not_invariant
#print axioms Article93.zero_transmission_balance
#print axioms Article93.SSH_four
#print axioms Article93.SSH_six
#print axioms Article93.contacts_six
```

```
#print axioms Article93.histogram_zero
#print axioms Article93.histogram_one
#print axioms Article93.histogram_two
#print axioms Article93.histogram_three
#print axioms Article93.histogram_total
#print axioms Article93.reversal_1_contacts
#print axioms Article93.bound_1_contacts
#print axioms Article93.reversal_3_contacts
#print axioms Article93.bound_3_contacts
#print axioms Article93.reversal_5_contacts
#print axioms Article93.bound_5_contacts
#print axioms Article93.reversal_7_contacts
#print axioms Article93.bound_7_contacts
#print axioms Article93.reversal_9_contacts
#print axioms Article93.bound_9_contacts
#print axioms Article93.six_site_mean_crossmultiply
#print axioms Article93.imperfect_example
```