The Complete Picture — Coq Formalization and Commentary

Packaged theorems for nested hypergraphs, weighted tensors, dynamics, and universal connectivity

```
The complete picture (ASCII version):

For all n >= 1 and all x1,...,xn : U,
  if R_n(x1,...,xn) then there exist
    NG : NestedGraph, w : R, t : Time
  such that:
    - (x1,...,xn) ∈ hyperedges(outer_graph(NG))  [set membership; see code]
    - NestedWeightedTensor(NG, x1,...,xn, t) = w
    - There exists f : NestedGraph × Time -> NestedGraph with
      DynamicPreservation(f, NG, t, R_n)
    - For all x : U, there exist m >= 1 and y1,...,y_{m-1} : U such that
      R_m(x, y1,...,y_{m-1}).
```

Coq Source (Complete_Picture.v)

```
(*
  Complete_Picture.v
  ==================
  Two variants of the "Complete Picture" packaging theorems.
  1) LIST-ARITY VERSION (matches your working code)
  2) VECTOR-ARITY VERSION (type-safe arity, uses List.In explicitly to avoid clash)
*)

From Coq Require Import List Arith PeanoNat.
Import ListNotations.

(* ========================================================= *)
(* =============== 1) LIST-ARITY VERSION ================== *)
(* ========================================================= *)

Section ListArity.

  Parameter U : Type.
  Definition Hyperedge := list U.

  Record Graph := { hedges : list Hyperedge }.

  Record NestedGraph := {
    outer_graph : Graph;
    inner_graph : Hyperedge -> option Graph
  }.

  Parameter Time Weight : Type.
  Parameter NestedWeightedTensor : NestedGraph -> Hyperedge -> Time -> Weight.

  Definition Evolution := NestedGraph -> Time -> NestedGraph.
  Definition NaryRelation (n:nat) := Hyperedge -> Prop.

  Definition DynamicPreservation
    (n:nat) (Rel : NaryRelation n) (f:Evolution) (NG:NestedGraph) (t:Time) : Prop :=
    forall e, Rel e -> In e (hedges (outer_graph NG))
          -> In e (hedges (outer_graph (f NG t))).

  Axiom relation_implies_structure :
    forall (n:nat) (Rel:NaryRelation n) (xs:Hyperedge),
```

```
    n > 0 -> length xs = n -> Rel xs ->
    exists (NG:NestedGraph) (w:Weight) (t:Time),
      In xs (hedges (outer_graph NG))
      /\ NestedWeightedTensor NG xs t = w.

Axiom structure_implies_dynamics :
  forall (n:nat) (Rel:NaryRelation n) (xs:Hyperedge) (NG:NestedGraph) (t:Time),
    Rel xs ->
    In xs (hedges (outer_graph NG)) ->
    exists (f:Evolution), DynamicPreservation n Rel f NG t.

Axiom universal_connectivity :
  forall (x:U),
    exists (m:nat) (Relm:NaryRelation m) (ys:Hyperedge),
      m > 0 /\ length ys = m /\ In x ys /\ Relm ys.

Theorem Complete_Picture :
  forall (n:nat) (Rel:NaryRelation n) (xs:Hyperedge),
    n > 0 -> length xs = n -> Rel xs ->
    (exists (NG:NestedGraph) (w:Weight) (t:Time),
        In xs (hedges (outer_graph NG))
     /\ NestedWeightedTensor NG xs t = w)
    /\ (exists (NG:NestedGraph) (t:Time),
            In xs (hedges (outer_graph NG))
         -> exists (f:Evolution), DynamicPreservation n Rel f NG t)
    /\ (forall x:U, exists (m:nat) (Relm:NaryRelation m) (ys:Hyperedge),
            m > 0 /\ length ys = m /\ In x ys /\ Relm ys).
Proof.
  intros n Rel xs Hn Hlen HRel.
  destruct (relation_implies_structure n Rel xs Hn Hlen HRel)
    as [NG [w [t [Hin Hwt]]]].
  assert (Hdyn_pack :
    exists NG' t',
      In xs (hedges (outer_graph NG')) ->
      exists f, DynamicPreservation n Rel f NG' t').
  { exists NG, t. intro Hin'.
    destruct (structure_implies_dynamics n Rel xs NG t HRel Hin') as [f Hpres].
    now exists f. }
  split.
  - now exists NG, w, t.
  - split.
    + exact Hdyn_pack.
    + intro x. apply universal_connectivity.
Qed.

Theorem Complete_Picture_strong :
  forall (n:nat) (Rel:NaryRelation n) (xs:Hyperedge),
    n > 0 -> length xs = n -> Rel xs ->
    (exists (NG:NestedGraph) (w:Weight) (t:Time) (f:Evolution),
        In xs (hedges (outer_graph NG))
     /\ NestedWeightedTensor NG xs t = w
     /\ DynamicPreservation n Rel f NG t)
    /\ (forall x:U, exists (m:nat) (Relm:NaryRelation m) (ys:Hyperedge),
            m > 0 /\ length ys = m /\ In x ys /\ Relm ys).
Proof.
  intros n Rel xs Hn Hlen HRel.
  destruct (relation_implies_structure n Rel xs Hn Hlen HRel)
    as [NG [w [t [Hin Hwt]]]].
  destruct (structure_implies_dynamics n Rel xs NG t HRel Hin)
    as [f Hpres].
  split.
  - exists NG, w, t, f. repeat split; assumption.
  - intro x. apply universal_connectivity.
Qed.

Corollary Complete_Picture_binary :
  forall (Rel2:NaryRelation 2) (xy:Hyperedge),
    length xy = 2 -> Rel2 xy ->
    exists NG w t f,
      In xy (hedges (outer_graph NG))
```

```
      /\ NestedWeightedTensor NG xy t = w
      /\ DynamicPreservation 2 Rel2 f NG t.
  Proof.
    intros Rel2 xy Hlen Hrel.
    assert (Hpos : 2 > 0) by exact (Nat.lt_0_succ 1).
    destruct (Complete_Picture_strong 2 Rel2 xy Hpos Hlen Hrel) as [H _].
    exact H.
  Qed.

End ListArity.

(* ========================================================= *)
(* ============== 2) VECTOR-ARITY VERSION ================== *)
(* ========================================================= *)

From Coq Require Import Vectors.Vector.
Import VectorNotations.

Section VectorArity.

  Parameter UV : Type.

  Definition HEdge (n:nat) := Vector.t UV n.
  Definition SigEdge := { n : nat & HEdge n }.

  Record GraphV := { hedgesV : list SigEdge }.

  Record NestedGraphV := {
    outer_graphV : GraphV;
    inner_graphV : SigEdge -> option GraphV
  }.

  Parameter TimeV WeightV : Type.
  Parameter NestedWeightedTensorV : NestedGraphV -> SigEdge -> TimeV -> WeightV.

  Definition EvolutionV := NestedGraphV -> TimeV -> NestedGraphV.
  Definition NaryRelV (n:nat) := HEdge n -> Prop.

  (* IMPORTANT: use List.In to avoid the vector In/arity mismatch *)
  Definition DynamicPreservationV
    (n:nat) (Rel:NaryRelV n) (f:EvolutionV) (NG:NestedGraphV) (t:TimeV) : Prop :=
    forall (e:HEdge n),
      Rel e ->
      List.In (existT _ n e) (hedgesV (outer_graphV NG)) ->
      List.In (existT _ n e) (hedgesV (outer_graphV (f NG t))).

  Axiom relation_implies_structureV :
    forall (n:nat) (Rel:NaryRelV n) (e:HEdge n),
      n > 0 -> Rel e ->
      exists (NG:NestedGraphV) (w:WeightV) (t:TimeV),
        List.In (existT _ n e) (hedgesV (outer_graphV NG))
        /\ NestedWeightedTensorV NG (existT _ n e) t = w.

  Axiom structure_implies_dynamicsV :
    forall (n:nat) (Rel:NaryRelV n) (e:HEdge n) (NG:NestedGraphV) (t:TimeV),
      Rel e ->
      List.In (existT _ n e) (hedgesV (outer_graphV NG)) ->
      exists (f:EvolutionV), DynamicPreservationV n Rel f NG t.

  Axiom universal_connectivityV :
    forall (x:UV),
      exists (m:nat) (Relm:NaryRelV m) (e:HEdge m),
        m > 0
        /\ List.In x (Vector.to_list e)
        /\ Relm e.

  Theorem Complete_Picture_V :
    forall (n:nat) (Rel:NaryRelV n) (e:HEdge n),
      n > 0 -> Rel e ->
      (exists (NG:NestedGraphV) (w:WeightV) (t:TimeV),
```

```
          List.In (existT _ n e) (hedgesV (outer_graphV NG))
       /\ NestedWeightedTensorV NG (existT _ n e) t = w)
      /\ (exists (NG:NestedGraphV) (t:TimeV),
              List.In (existT _ n e) (hedgesV (outer_graphV NG))
           -> exists (f:EvolutionV), DynamicPreservationV n Rel f NG t)
      /\ (forall x:UV, exists (m:nat) (Relm:NaryRelV m) (e':HEdge m),
              m > 0 /\ List.In x (Vector.to_list e') /\ Relm e').
  Proof.
    intros n Rel e Hn HRel.
    destruct (relation_implies_structureV n Rel e Hn HRel)
      as [NG [w [t [Hin Hwt]]]].
    assert (Hdyn_pack :
      exists NG' t',
        List.In (existT _ n e) (hedgesV (outer_graphV NG')) ->
        exists f, DynamicPreservationV n Rel f NG' t').
    { exists NG, t. intro Hin'.
      destruct (structure_implies_dynamicsV n Rel e NG t HRel Hin') as [f Hpres].
      now exists f. }
    split.
    - now exists NG, w, t.
    - split.
      + exact Hdyn_pack.
      + intro x. apply universal_connectivityV.
  Qed.

  Theorem Complete_Picture_V_strong :
    forall (n:nat) (Rel:NaryRelV n) (e:HEdge n),
      n > 0 -> Rel e ->
      (exists (NG:NestedGraphV) (w:WeightV) (t:TimeV) (f:EvolutionV),
          List.In (existT _ n e) (hedgesV (outer_graphV NG))
       /\ NestedWeightedTensorV NG (existT _ n e) t = w
       /\ DynamicPreservationV n Rel f NG t)
      /\ (forall x:UV, exists (m:nat) (Relm:NaryRelV m) (e':HEdge m),
              m > 0 /\ List.In x (Vector.to_list e') /\ Relm e').
  Proof.
    intros n Rel e Hn HRel.
    destruct (relation_implies_structureV n Rel e Hn HRel)
      as [NG [w [t [Hin Hwt]]]].
    destruct (structure_implies_dynamicsV n Rel e NG t HRel Hin)
      as [f Hpres].
    split.
    - exists NG, w, t, f. repeat split; assumption.
    - intro x. apply universal_connectivityV.
  Qed.

End VectorArity.
```

What the Coq Code Proves

The Coq file proves a packaged theorem suite that turns relational truths into existential witnesses for structure, dynamics, and connectivity. Under three axioms, any valid n-ary relation Rel on a hyperedge implies:

• a NestedGraph NG with the hyperedge in its outer graph, a time t, and a weight w from the NestedWeightedTensor;
• an evolution f that preserves Rel-hyperedges (DynamicPreservation);
• universal participation of all entities in some relation.

The strong variants unify these under the same NG and t, and the vector form adds type-safety. Proofs destruct axioms to construct witnesses; the binary corollary specializes to n = 2.

Meaning of the Proof

The theorems provide a usable interface: from Rel e you obtain witnesses for topology (NG), annotation (w), evolution (f), and connectivity. Relations are operational—fit for reasoning in physics or social models. Lists support variable arity; vectors enforce arity at the type level. The strong form emphasizes coherence: structure and dynamics are packaged together, embodying "relation as the unit of reality."

Significance of the Proof

This suite acts as a contract for UCF/GUTT-style instantiations: given the axioms, any domain instantiation yields a coherent model. Dual arity demonstrates robustness and scalability. It synthesizes prior propositions into a machine-verifiable backbone, making the relational theory computable and suitable for invariants, simulations, and extraction to programs.

Implications of the Proof

• Representational completeness: true relations embed concretely in nested hypergraphs.
• Dynamical adequacy: preservation enables stability metrics (Φ) and forecasting.
• No isolation: global connectivity guarantees every entity participates.
• Practical workflow: destruct witnesses for proofs and simulations; instantiate U, Rel_n, Weight, Time, and f for concrete domains (atoms/bonds, agents/trust, particles/interactions).
• Multi-scale modeling: inner graphs carry mechanisms/contexts; outer graphs capture macro ties.

Author's Note

"I've already test driven this engine... which is to be expected... I'm happy with the ride."