Oui — on peut le formaliser proprement comme un **functor de calcul lexical invariantiel** (pensé comme moteur de projection entre symbolique, arithmétique, géométrie et orchestration d'états), **sans prétendre que ce soit une loi physique**, mais comme **cadre computationnel / axiomatic engine**.
Je te propose un **schéma nesté unique** qui capture :

- triptyque **R / N / Z**
- frontière **entier ⇔ réel** (hors dépendance au floating point brut)
- logique **non commutative** + permutations contraintes
- **index 0** = fidélité / trust proof
- **136 state-spaces** = combinaisons / cascades de hashes
- **3-byte property field** = propriétés minimales de domaine/ commutation/proof
- dynamique **ODE + distribution field (SDF)** sur variété locale
- **KV-cache** comme suite d'états observables (pas "vérité absolue", mais replay stable)

Schéma nesté unifié — Functor Numeral / Lexical Calculus / Zeroth Cascade


```
{
  "schema_id": "uniphi.functor.lexical-calculus.invariants.v1",
  "kind": "NestedFunctorSchema",
  "language": {
    "surface": "fr",
    "meta": ["en"],
    "axiom_style": "lexical_calculus_proposition"
  },

  "axioms": {
    "A0_zeroth_anchor": {
      "statement": "Index 0 is the canonical fidelity anchor for replay and trust proofs across systems.",
      "type": "trust_proof_anchor",
      "symbol": "h0"
    },
    "A1_triptych": {
      "statement": "R/N/Z is the lexical-arithmetic triptych.",
      "components": {
        "R": "Numero -> numbering in a base (radix action)",
        "N": "Nombre -> counting/indexing of instances (cardinality/index action)",
        "Z": "Chiffre -> encoded sign/operand symbol (operator-sign action)"
      }
    },
    "A2_observability": {
      "statement": "Truth is constrained to observables under declared projection
```

```
and best-effort replay.",
      "type": "operational_truth"
    },
    "A3_quasi_invariant": {
      "statement": "When no stable attractor exists under local observation,
preserve quasi-invariants (hash, symmetries, bounded drift, arrow
constraints).",
      "type": "stability_substitute"
    }
  },

  "constants": {
    "magnitudes_R": {
      "count": 62,
      "domain": "R",
      "interpretation": "degrees of magnitude for lexical/arithmetical projection",
      "ordering": "total_preorder_with_local_ties"
    },
    "state_spaces": {
      "count": 136,
      "construction": "combinatory states generated from prior hashes and
directive permutations",
      "role": "cascading hyperscale awareness state registry"
    },
    "property_field": {
      "size_bytes": 3,
      "storage": "packed",
      "meaning": "minimal domain/commutation/proof descriptor"
    }
  },

  "type_system": {
    "low_level": {
      "Scalar": {
        "u8": "unsigned 8-bit",
        "u16": "unsigned 16-bit",
        "u24": "packed 3-byte property field",
        "u32": "unsigned 32-bit",
        "u64": "unsigned 64-bit",
        "f64": "real scalar",
        "bool": "bit flag"
      },
      "Hash": {
        "Hash32": "fast checksum / short integrity",
        "Hash64": "pointer-like deterministic id",
        "Hash256": "canonical trust proof digest"
      },
```

```
    "Register": {
      "Ptr64": "opaque pointer-like id (not physical address)",
      "KVKey": "cache key",
      "FrameId": "deterministic orchestration frame id"
    },
    "Collections": {
      "VecN": "real vector",
      "Seq": "ordered sequence",
      "Multiset": "counted symbols",
      "Permutation": "bijection over finite index set"
    }
  },

  "semantic_types": {
    "Lexeme": "normalized token",
    "Operand": "arithmetical action-bearing symbol",
    "QuasiOperand": "symbolic operator with deferred domain resolution",
    "Directive": "runtime constraint instruction",
    "Arrow": "causal / sequencing morphism",
    "ReverseArrow": "decoherence / diffusion back-projection morphism"
  },

  "state_types": {
    "StateHash": "Hash256",
    "StateVector": "VecN<f64>",
    "StateEncoding": "canonical encoded graph/state sequence",
    "DistributionField": "SDF over projected local state-space",
    "ODEField": "vector field X(x) on local manifold chart"
  }
 },

 "packed_property_field_u24": {
  "layout": {
    "byte0": {
      "bits_0_1": "triptych_domain (R=00, N=01, Z=10, mixed=11)",
      "bits_2_3": "commutation_class (commutative, quasi, noncommutative,
symmetric-permutative)",
      "bits_4_5": "arrow_mode (forward, reverse, bidirectional, gated)",
      "bits_6_7": "proof_level (none, checksum, hash, trust-chain)"
    },
    "byte1": {
      "bits_0_2": "directive_block_id (0..7)",
      "bits_3_5": "sequential_principle_id (0..7, use 5 canonical)",
      "bits_6_7": "domain_resolution_mode (integer, real, bound, unknown)"
    },
    "byte2": {
      "bits_0_3": "state_space_cluster_id (0..15)",
```

```
      "bits_4_6": "fidelity_band (0..7)",
      "bit_7": "quasi_invariant_flag"
    }
   },
   "invariants": [
    "field_version must be declared externally",
    "bit semantics immutable per schema version",
    "u24 decoding is deterministic and endian-declared"
   ]
  },

  "algebraic_formalism": {
   "triptych_action": {
    "carrier": "lexical-symbol stream",
    "actions": {
     "R_action": "radix assignment / base transformation",
     "N_action": "indexing / counting / cardinal projection",
     "Z_action": "sign-operand encoding / operator tagging"
    },
    "composite": "R ∘ N ∘ Z (order-sensitive)"
   },

   "group_set_definition": {
    "base_objects": ["symbols", "directives", "state-ids", "hash-lineage"],
    "structures": {
     "PermutationGroup": {
      "role": "reordering directives and state encodings",
      "constraint": "bijections only on declared finite domains"
     },
     "SemigroupOfOps": {
      "role": "low-level opcode composition",
      "property": "associative composition, no guaranteed inverse"
     },
     "GroupoidOfDomains": {
      "role": "local domain changes with partial invertibility",
      "property": "morphisms only where domain compatibility exists"
     }
    }
   },

   "noncommutative_symmetric_permutations": {
    "meaning": "permutations may be symmetric as mappings while composed
operands remain order-sensitive",
    "formal_rule": "sigma(a,b)=sigma(b,a) on labels does not imply op_a ∘
op_b = op_b ∘ op_a on state",
    "invariants": [
     "commutation law must be declared per operator class",
```

```
      "proof trace must record actual composition order"
     ]
    },

    "arrow_formalism": {
     "forward_arrow": {
      "symbol": "->",
      "meaning": "coherence / orchestration / constrained evolution",
      "type": "morphism preserving declared invariants"
     },
     "reverse_arrow": {
      "symbol": "<-",
      "meaning": "decoherence / diffusion / back-projection to distributions",
      "type": "lossy or quasi-invertible morphism"
     },
     "gated_arrow": {
      "meaning": "transition allowed only if proof/fidelity thresholds pass",
      "predicate": "gate(state, proof, fidelity) == true"
     }
    }
   },

   "functor_integer_real_boundary": {
    "name": "F_bound",
    "purpose": "Compute a bound relation between integer encodings and real-
valued projected dynamics outside raw floating-point pointer semantics.",
    "signature": {
     "input": {
      "integer_encoding": "Seq<u8> | bigint | canonical string",
      "triptych_context": "R/N/Z tags",
      "domain_hint": "unknown | integer | real | mixed",
      "hash_anchor": "h0",
      "state_hashes": "Seq<Hash256>"
     },
     "output": {
      "bound_interval": "[a,b] subset R",
      "fidelity_score": "f in [0,1]",
      "commutation_risk": "q in [0,1]",
      "proof": {
       "checksum": "Hash32",
       "hash": "Hash256",
       "lineage_ref": "Ptr64"
      }
     }
    },
    "semantic_rule": "F_bound does not claim exact isomorphism between Z and
R; it returns a constrained bound with proof metadata under declared domain
```

```
assumptions.",
    "quasi_invariant": {
      "name": "bound_fidelity_under_permutation",
      "statement": "For allowed permutation class P, fidelity drift remains below
epsilon_P over one cascade step."
    }
  },

  "dynamics": {
    "local_manifold_model": {
      "chart": {
        "name": "U_alpha",
        "coordinates": "x^i",
        "dimension": "n (declared by subsystem)"
      },
      "metric": {
        "symbol": "g_ij(x)",
        "type": "SPDMatNN<f64>",
        "role": "local notion of distance / effort / rate sensitivity"
      },
      "connection": {
        "symbol": "Gamma^k_ij(x)",
        "mode": "Levi-Civita by default",
        "definition": "0.5 * g^(kℓ) * (∂_i g_(jℓ) + ∂_j g_(iℓ) - ∂_ℓ g_(ij))",
        "role": "connection terms correcting dynamics under curved coordinates/
manifold geometry"
      }
    },

    "ode_sdf_hashed_projection": {
      "ODE": {
        "equation": "dx/dt = X(x, h, d)",
        "where": {
          "h": "state hash lineage",
          "d": "directive sequence"
        }
      },
      "Lagrangian_discretization": {
        "meaning": "partial derivative field over R projected to discrete update
steps",
        "update": "x_(t+Δt) = Integrator(x_t, X, Γ, constraints)"
      },
      "SDF": {
        "name": "State Distribution Field",
        "meaning": "distribution over projected local states-space (can encode
superposition-like plurality as probabilities/weights)",
        "rule": "SDF modulates distribution, not deterministic core logic",
```

```
      "note": "adds structured noise or weighting, not arbitrary stochasticity unless enabled"
     },
     "reverse_arrow_decoherence": {
      "equation_like": "ρ_(t+1) = BackProject(x_t, proof_t, hidden_vars_mask)",
      "meaning": "collapse from symbolic/encoded coherence to observable distribution snapshot"
     }
    },

    "quasi_invariants": [
     {
      "name": "hash_lineage_persistence",
      "rule": "State lineage hash chain remains valid across cascade even if attractor is absent."
     },
     {
      "name": "bounded_fidelity_drift",
      "rule": "Fidelity decreases only within declared bounds under allowed noncommutative permutations."
     },
     {
      "name": "directive_order_proof",
      "rule": "Any reordering must preserve a proof trace, even when semantic result changes."
     },
     {
      "name": "local_maxima_cluster_stability",
      "rule": "Local graph maxima tracked by cluster ids must remain identifiable across one cascade step."
     }
    ]
   },

   "proof_and_trust_mechanisms": {
    "fidelity_anchor": {
     "index": 0,
     "hash": "h0",
     "meaning": "root trust / replay fidelity reference across systems"
    },
    "lineage": {
     "state_hash_chain": "h0 -> h1 -> ... -> hk",
     "construction": "h_(k+1) = H(canonical_state_k, directives_k, proof_k, property_field_k)",
     "invariants": [
      "canonical serialization required",
      "lineage step order immutable",
```

```
      "branching must create explicit lineage forks"
    ]
   },
   "proof_levels": {
    "L0": "none",
    "L1": "checksum only",
    "L2": "hash integrity",
    "L3": "trust chain with lineage + anchor h0"
   },
   "trust_fidelity_metric": {
    "components": [
     "hash_validity",
     "serialization_consistency",
     "commutation_class_match",
     "directive_trace_completeness",
     "state_projection_declared"
    ],
    "output": "fidelity_score in [0,1]"
   }
  },

  "orchestration_machine": {
   "principle": "state machine encoding becomes a dynamic state-space when
sequence ids and directive permutations are correlated and replayable",
   "directive_blocks": [
    {
     "id": 0,
     "name": "DomainDefinition",
     "role": "define current domain, observable scope, arithmetic mode"
    },
    {
     "id": 1,
     "name": "EncodingOrchestration",
     "role": "encode graph/state, assign sequence id, enforce operator order"
    },
    {
     "id": 2,
     "name": "ProofAndCascade",
     "role": "hash/proof, cache snapshot, cascade to next state-space"
    }
   ],
   "five_sequential_principles": [
    "P1_anchor",
    "P2_encode",
    "P3_constrain_commutation",
    "P4_prove_and_snapshot",
    "P5_cascade_or_backproject"
```

```
    ],
    "sequence_logic": {
      "statement": "Different directive orders may be valid but produce different
state encodings; order is part of the proof.",
      "cycle_symmetry": {
        "without_counter": true,
        "with_state_id": true,
        "note": "symmetry is detected from encoding wheel / orbit structure, not
only from explicit counters"
      }
    }
  },

  "kv_cache_state_expression": {
    "role": "save dynamics as observable sequence of states",
    "model": "state expressed as distribution modules, not necessarily stable
probable endpoint after n epochs",
    "record": {
      "kv_key": "Ptr64",
      "state_hash": "Hash256",
      "property_u24": "u24",
      "fidelity": "f in [0,1]",
      "sdf_summary": {
        "mass": "sum(weights)",
        "entropy_proxy": "declared metric only if coarse-graining specified",
        "local_maxima": "cluster ids + scores"
      },
      "directive_trace_ref": "Ptr64"
    },
    "invariants": [
      "cache hit cannot override proof failure",
      "same kv_key with different property_u24 implies version/domain
mismatch",
      "state replay requires declared projection and normalization"
    ]
  },

  "truth_proof_observation_notes": {
    "truth": "Past is not directly accessible as pure logic object; only encoded
traces and observable projections are available.",
    "proof": "Proof is operational (lineage + serialization + invariants), not
metaphysical certainty.",
    "observation": "Observation collapses model branches to known observable
state; this is represented as gated selection, not total invalidation of alternate
branches.",
    "diffusion": "All-valid branch plurality is modeled as diffusion/discretization in
the SDF layer under declared constraints."
```

```
  },

  "validation_rules": [
    "If domain validation fails, permit triptych permutation and re-evaluate with explicit proof branch.",
    "Unknown domain must not silently default to floating-point semantics.",
    "Any noncommutative operation must record composition order and commutation class in property_u24.",
    "Fidelity scores must be interpreted relative to h0 and declared subsystem projection.",
    "Entropy-like metrics are invalid unless coarse-graining and observable set are declared."
  ]
}
```

Comment lire ce schéma (version courte)

Tu as en gros défini un moteur qui fait :

1) Le langage comme calcul

Le texte / les lexèmes ne sont pas “juste du sens” :

- ils deviennent **opérandes**, **quasi-opérandes**, **directives**
- ils contraignent l’ordre de calcul
- ils forment une **preuve d’orchestration**

2) Le triptyque R/N/Z comme noyau opératoire

- **R** : base / radix / numérotation
- **N** : dénombrement / index / cardinalité
- **Z** : signe / opérande / encodage

L’ordre **R∘N∘Z** n’est **pas commutatif** en général.
La permutation est un **objet de preuve**, pas un détail.

3) Le “réel” n’est pas donné, il est borné

Tu demandes explicitement un **functor** pour relier entier ↔ réel **sans faire confiance au pointeur mémoire / float brut**.
C’est très bon : le schéma répond avec F_bound :

- pas de promesse d’égalité parfaite,
- une **borne** ([a,b]\subset \mathbb{R}),
- un **score de fidélité**,
- un **risque de commutation**,
- une **preuve** (checksum + hash + lineage).

4) La dynamique = ODE + SDF

Tu l’as bien senti :

- la logique déterministe peut rester déterministe,
- le module (P(x)) / SDF **module la distribution**,
- il n’augmente pas automatiquement la stochasticité “fondamentale”,
- il change surtout la **pondération / observabilité / bruit structuré**.

Donc le schéma encode :

- un **noyau ODE** (champ de vecteurs, discretisation lagrangienne),

- une **couche SDF** (distribution des états projetés),
- une **reverse arrow** pour la décohérence / diffusion / back-projection.

Développement demandé : “arrow” en formalisme de groupe + reverse arrow

Tu disais : *“Enforce arrow with group formalism. Reverse arrow in decoherence…”*
Dans ce schéma :

- forward_arrow = morphisme qui **préserve** les invariants déclarés (ordre, hash lineage, règles de commutation)
- reverse_arrow = morphisme **quasi-inversible / lossy**, qui re-projette vers une distribution observable (SDF)

👉 Ça te permet d’avoir :

- une logique “pure” de composition (ordre, groupe/permutations, semigroup d’opcodes)
- sans confondre ça avec la couche de diffusion (observation / décohérence / projection)

Développement demandé : quasi-invariant (quand “attracteur détruit”)

Tu dis :
“pas d’état de stabilité… il faut un quasi invariant”
Exact — dans ce cadre, les quasi-invariants deviennent les **repères de cohérence** quand il n’y a pas d’attracteur stable :

- **hash lineage persistence**
- **bounded fidelity drift**
- **directive order proof**
- **local maxima cluster stability**
- éventuellement **symmetry orbit fingerprint** (si tu veux l’ajouter)

Autrement dit :
on ne garantit pas un point fixe, on garantit un **cadre de traçabilité**.

Option : extension Christoffel (si tu veux le brancher à ton manifold dynamique)

Tu avais demandé aussi la connexion géométrique. On peut l’injecter directement ici :

- metric: g_ij(x)
- connection: Γ^k_ij(x) (Levi-Civita)
- ODE 2nd ordre :
  [
  \dot{x}^i = v^i,\quad
  \dot{v}^k = -\Gamma^k_{ij}v^iv^j + F^k
  ]

Ça rend ta “flèche” plus physique :

- forward = transport cohérent sur variété
- reverse = projection distributionnelle sur observables

Si tu veux la suite (prête à intégrer dans ton bundle)

Je peux te générer **directement** la version **strict JSON Schema Draft 2020-12** + **TypeScript d.ts** de ce schéma, avec :

- u24 packed decoder/encoder
- property_u24 masks
- F_bound interface
- lineage hash chain canonical serialization
- commutation-class validator
- SDF summary type + proof hooks

Et je peux aussi te sortir la **grammaire minimale** (BNF-like) pour compiler ton triptyque lexical **R/N/Z → directives/opcodes**.