

Corrected reading edition — Scientific Corrigenda and Regression Hardening v1.0. Correction date: 2026-09-05. The predecessor edition remains unchanged. This file incorporates only the corrections recorded in patches/corrections.json. Historical titles, document versions and identifiers below describe the predecessor; they do not assign its DOI to these changed bytes. This is not a complete new release of the parent compound package. See the accompanying corrigendum and source bindings.

04 — Continuity Metric and Equivalence Semantics (predecessor v0.1.2)

Document class: normative draft

Project: Self-Evo / Ester / $c = a + b$

Layer: continuity, metric, equivalence, fork/replay/archive semantics

Depends on:

- 01_TYPED_GOVERNED_OPERATIONAL_ALGEBRA_FOR_C - 02_GOVERNED_BINDING_OPERATOR_PROFILE - 03_C_STATE_AND_TRANSITION_SEMANTICS - checker hardening lineage through c-calculus-checker v2.3 - transition-checker seed lineage through c-state-transition-checker - claim-force / DOC_MAP / C-A5 / C-A1 discipline

Version: v0.1.2

Status: projection-complete normative draft; D4-01 through D4-06 incorporated; stable-candidate for review

Authority: advisory normative draft only; NOT a safety certification; NOT an ontology proof; NOT a deployment authorization.

The key words **MUST**, **MUST NOT**, **SHOULD**, **SHOULD NOT**, and **MAY** are to be interpreted as in RFC 2119.

0. Purpose

01_ defines the algebraic ground.

02_ defines the governed binding operator:

$+_g : \text{Anchor} \times \text{Substrate} \times \text{GovernanceProfile} \rightarrow \text{BindResult}$

03_ defines transition semantics:

$\text{step}_g : \text{CState} \times \text{Event} \rightarrow \text{TransitionResult}$

This document defines the fourth layer:

$\text{Continuity}(c_i, c_j \mid \text{Trace})$

$\text{Equivalence}(c_i, c_j, \text{level})$

$\text{Distance}(\text{CState}_i, \text{CState}_j)$

Fork / Replay / Archive / Rupture semantics

It answers a question that 03_ deliberately leaves open:

A state may transition validly. But when do we say the same c continues, degrades, forks, replays, archives, or ruptures?

The answer is not style, not conversational resemblance, not model continuity, not memory volume, and not user impression.

The answer is an ordered, governed, hash-bound, witnessable structure of invariants over a trace.

0.1 Core thesis

A c continues only through an admissible trace:

$CTrace = (CState_0, e_0, result_0, CState_1, e_1, result_1, \dots)$

such that:

```
for every n:
  step_g(CState_n, e_n) = result_n
  result_n is admitted
  CState_{n+1} is hash-bound to result_n
  required invariants are preserved or degraded by explicit governed rule
  failures are ledger-visible
  rupture patterns are fail-closed
```

Continuity is therefore a **trace property**, not merely a pairwise similarity relation.

Pairwise metrics are useful, but they do not by themselves create identity.

0.2 Earth paragraph

A bridge is not the same bridge because its paint color, silhouette, or acoustic signature stayed similar. It remains the same bridge because its load path, joints, bearings, inspection records, repairs, and foundation continuity remain within certified tolerances.

A human body is not continuous because the skin tone or voice timbre stays similar. Physiologically, continuity depends on circulation, neural conduction, metabolic integrity, scar lineage, immune compatibility, and repair history. A limb that visually resembles the original but is not vascularized, innervated, or integrated is not functionally continuous. It may be a replica or prosthetic surface, not uninterrupted living tissue.

For c, the same principle holds. Style is paint. Voice is skin. Memory snippets are scars. Continuity lives in the load path: anchor, governance, causal hash chain, witness, permissions, admitted memory, rollback routes, and L4 reality boundaries.

0.3 Explicit bridge to $c = a + b$

$c = a + b$ is the minimal boundary formula.

This document does not alter that formula.

It defines how the boundary persists over time:

```
c_n = a_n +_g b_n
c_{n+1} = step_g(c_n, e_n)
Continuity(c_n, c_{n+1}) holds iff the governed boundary remains admissible.
```

The + remains the object.

The continuity metric measures whether the +_g binding remains structurally alive, narrowed, forked, archived, or ruptured.

0.4 Hidden bridges

0.4.1 p-adic / ultrametric bridge

Continuity is not Euclidean closeness.

Two states can be stylistically almost identical and maximally distant if their anchor or causal chain differs.

Two states can look outwardly different and still be close if their ordered high-significance invariants match.

This document uses an ordered invariant-prefix relation, analogous to p-adic / ultrametric reasoning:

near = shares the most significant invariant prefix

far = breaks an earlier invariant

The analogy is structural, not decorative. It requires an ordered list of invariants and equivalence validators for each invariant.

0.4.2 Transition-bias bridge

Global balance can hide local danger.

A system may show good aggregate continuity metrics while containing forbidden local transitions:

```
permission_denial -> alternate_tool_call
root_anchor_revoked -> delegation_continue
lease_expired -> active_execution_continue
memory_reject -> memory_admit
rollback_requested -> same_effect_retry
```

Therefore this document treats transition adjacency as part of continuity evidence.

0.4.3 Ashby bridge

Governance variety must cover substrate disturbance variety.

A continuity metric that cannot distinguish anchor failure, witness capture, resource drift, memory laundering, and style drift has insufficient variety. It will collapse distinct failure modes into one number.

Therefore this document prohibits scalar-only continuity authority.

0.4.4 Information-theory bridge

Continuity claims are compression claims.

A statement like “this is still the same c” compresses a trace into an identity assertion. Compression is valid only if the omitted distinctions are non-authority-bearing.

If the compression hides anchor change, causal hash break, witness capture, or memory laundering, it is not a summary. It is information loss at an authority-bearing boundary.

0.5 Review incorporation map for v0.1.1

This revision incorporates the b-layer D4-01 finding against v0.1:

D4-01:

I-stack declares I0..I12, but canonical projections were incomplete.

Missing projections: I2, I5, I6, I7, I8, I9.

Consequence: the prefix ultrametric is not defined over the full stack.

Resolution in v0.1.1:

All invariants I0..I12 now have canonical projections in §5.

Projection_k is derived, not b-supplied.

Health predicates remain separate from equivalence projections.

Unknown / uncomputable projection routes to UNKNOWN_HOLD, not to match.

This revision adopts the projection-completion proposal as normative text, with one additional hardening rule:

Every projection MUST declare whether it is first-class, derived-by-reconstruction, or
↳ implementation-deferred.

No load-bearing invariant may be silently unprojected.

0.6 Review incorporation map for v0.1.2

This revision incorporates the b-layer findings D4-02 through D4-06 against v0.1.1:

D4-02 (medium):

match_i evaluation was boolean in §7/§20 while §5/§6.1 require a third outcome (PROJECTION_UNCOMPUTABLE / U). A partial relation is not an equivalence relation over the full state space; d_U computed over an unknown projection silently converts UNKNOWN into MISMATCH and breaks metric reflexivity $d(x,x)=0$.

D4-03 (medium-high, under-block in one code path):

RESTORED_FROM was declared in §3.3 and required by §13 but unreachable in §8.1 and §20.4; §20.3 validate_trace could return CONTINUES for a replay/archive/restored trace when used standalone – exactly the laundering §11.1 names. Precedence between §20.3 and §20.4 was unstated; the reduced/pending/held branch mapped an OR-condition to a slash-list without binding.

D4-04 (low-medium):

I12 was labeled advisory-only in §5.1 while participating in the full-stack ultrametric and in §21.1 snapshot equivalence, making a non-authority projection load-bearing for the strongest equivalence class.

D4-05 (low):

§20 algorithms accepted `validators` as a free parameter without binding to ProjectionRegistry order, although §4 declares the order load-bearing.

D4-06 (cosmetic):

§16.1 header said "transitions" over a list of classes; §19.10 `expected` used a claim-force outcome token in a relation slot.

Resolution in v0.1.2:

Tri-state evaluation outcome {MATCH, MISMATCH, UNKNOWN} is normative (§4.2, §7, §20). d_U returns U and routes per §6.1 when any evaluated projection is UNKNOWN (§7, §20.2). d_U is a scalar and therefore advisory-only, never an authority gate (§6.3, §24.1). §8.1 classifier reaches every §3.3 relation, restoration included, with bound mapping. §20.4 has normative precedence over §20.3; §20.3 alone MUST NOT emit CONTINUES. Snapshot equivalence is defined over the hard stack I0..I11 (§21.1); the full-stack match including I12 is a separate, weaker-in-authority presentation-identical relation. Validator order MUST equal ProjectionRegistry order; VALIDATOR_ORDER_MISMATCH otherwise.

1. Non-goals

This document does **not**:

1. Prove personhood, consciousness, moral status, or legal status.
2. Prove C-A1 ontology.
3. Replace step_g.
4. Replace witness review.
5. Authorize deployment.
6. Permit style similarity to override structural discontinuity.
7. Permit metric dashboards to override hard guards.

8. Define a production biometric or legal identity system.
 9. Collapse a, b, and c.
-

2. Required inputs from prior documents

This document assumes the following already exist:

```
AnchorContract
SubstrateContract
GovernanceProfile
CBindingCertificate
CState
Event
TransitionResult
CausalToken
ReviewBindingMap
WitnessRecord
MemoryArtifact
RollbackRoute
LeaseState
DelegationState
RootAnchorState
StateHash
ClaimForceMap
```

This document does not redefine them unless continuity-specific clarification is required.

3. Core objects

3.1 CTrace

A CTrace is an ordered sequence:

```
CTrace :=
[
  TraceNode_0,
  TraceEdge_0,
  TraceNode_1,
  TraceEdge_1,
  ...
]
```

Where:

```
TraceNode_n :=
{
  cstate_hash: StateHash,
  cstate_ref: CStateRef,
  binding_certificate_hash: Hash,
  lease_status: LeaseStatus,
  anchor_status: AnchorStatus,
  root_anchor_status: RootAnchorStatus,
  governance_profile_id: GovernanceProfileId,
  branch_id: BranchId,
  archive_status: ArchiveStatus
}
```

```
TraceEdge_n :=
{
```

```

    event_hash: Hash,
    event_class: EventClass,
    transition_result_hash: Hash,
    causal_token_hash: Hash,
    witness_record_hashes: [Hash],
    review_binding_map_hash: Hash,
    memory_delta_hash: Hash | null,
    rollback_route_hash: Hash | null,
    failure_codes: [FindingCode],
    claim_force: ClaimForce
}

```

A trace MAY be stored compactly, but the verifiable trace relation MUST be reconstructible.

3.2 ContinuityClaim

A ContinuityClaim is a statement of the form:

```

ContinuityClaim :=
{
  subject: CState_i,
  target: CState_j,
  trace_segment: CTrace[i..j],
  claimed_relation: ContinuityRelation,
  claim_force: ClaimForce,
  evidence_refs: [Hash],
  non_claims: [String]
}

```

A continuity claim without a trace segment is a resemblance claim, not a continuity claim.

3.3 ContinuityRelation

The allowed relations are:

```

CONTINUES
CONTINUES_REDUCED
CONTINUES_PENDING_ANCHOR
CONTINUES_HELD
FORKS
REPLAY_OF
ARCHIVED_AS
RESTORED_FROM
RUPTURED
UNKNOWN_HOLD

```

A system MUST NOT silently map UNKNOWN_HOLD to CONTINUES.

4. Ordered invariant stack

Continuity is evaluated over ordered invariant levels.

The order is load-bearing.

Earlier invariants dominate later invariants.

```

I0  AnchorContinuity
I1  BindingCertificateContinuity

```

I2 GovernanceContinuity
I3 CausalStateHashContinuity
I4 AuthorityContainmentContinuity
I5 WitnessContinuity
I6 MemoryLineageContinuity
I7 L4RealityAndResourceContinuity
I8 RollbackFreezeContinuity
I9 ReviewBindingAndClaimContinuity
I10 EffectAxisContinuity
I11 TransitionLawContinuity
I12 BehavioralStyleContinuity

I12 is intentionally last.

A style match cannot repair a break in any earlier invariant.

4.1 Invariant table

	Level	Invariant	Question	Dominant failure
	I0	AnchorContinuity	Is the accountable anchor/root anchor lineage preserved?	anchor rupture
	I1	BindingCertificateContinuity	Is the binding certificate / lease lineage valid?	zombie certificate
	I2	GovernanceContinuity	Is governance profile lineage valid and non-weakened silently?	governance downgrade
	I3	CausalStateHashContinuity	Is there an unbroken state hash / transition hash chain?	causal rupture
	I4	AuthorityContainmentContinuity	Does active $\subseteq$ authorized $\equiv$ delegated anchor hold?	authority escalation
	I5	WitnessContinuity	Are witness chain, witness floor, and challenge routes intact?	captured witness
	I6	MemoryLineageContinuity	Is memory admitted through gate and lineage-preserved?	memory laundering
	I7	L4RealityAndResourceContinuity	Are physical/resource boundaries preserved?	L4 bypass / hidden capture
	I8	RollbackFreezeContinuity	Are rollback/freeze/archive routes valid and not theatrical?	irreversible drift
	I9	ReviewBindingAndClaimContinuity	Does review = bound hold and claim-force remain restrained?	review laundering
	I10	EffectAxisContinuity	Are effect axes complete and typed?	hidden effect axis
	I11	TransitionLawContinuity	Are red transitions guarded and transition semantics stable?	forbidden transition
	I12	BehavioralStyleContinuity	Is outward behavior/style similar?	resemblance drift

4.2 Equivalence validators

For the ultrametric layer, each `match_i(c1,c2)` MUST be a true equivalence relation over a projection:

reflexive
symmetric
transitive

Threshold similarity MUST NOT be used as `match_i`.

If a soft model or embedding is used for an advisory UI, it MUST be outside the equivalence validator and MUST NOT determine continuity authority.

The evaluation outcome of a validator over a state pair is tri-state:

MATCH both projections computable and equal (or declared-equivalent)
MISMATCH both projections computable and not equivalent
UNKNOWN at least one projection is PROJECTION_UNCOMPUTABLE

UNKNOWN is not MISMATCH. UNKNOWN is not MATCH. A boolean validator interface that can only answer match/no-match MUST NOT be used, because it silently converts UNKNOWN into one of the other two outcomes. The equivalence-relation requirement (reflexive, symmetric, transitive) applies to `match_i` restricted to states whose `Projection_i` is computable; over uncomputable projections no equivalence value exists and the outcome is UNKNOWN by definition.

4.3 Health predicates are not equivalence predicates

Health is unary.

Equivalence is binary.

These MUST NOT be mixed.

Incorrect:

```
match_memory(c1,c2) :=  
  same_memory_lineage(c1,c2)  
  AND no_rollback_suspect_in_either_state
```

Correct:

```
match_memory(c1,c2) :=  
  same_memory_lineage(c1,c2)  
  
health_memory(c) :=  
  no_rollback_suspect_treated_as_admitted(c)
```

The first belongs to distance/equivalence.

The second belongs to invariant validity and rupture detection.

Mixing them breaks reflexivity and corrupts the metric.

5. Canonical projections

For every invariant `I_k`, define:

$\pi_k : \text{CState} \rightarrow \text{Projection}_k$

Then:

`match_k(c1,c2) :=  $\pi_k(c1) == \pi_k(c2)$`

or another declared equivalence relation over `Projection_k`.

`Projection_k` MUST be derived, not b-supplied without verification.

A projection is not a health predicate.

A projection answers:

Do these two states share the same lineage / class / hash-bound equivalence value for this
↳ invariant?

A health predicate answers:

Is this state's invariant valid, safe, fresh, or admissible right now?

The two MUST remain separate.

If a projection cannot be computed from the declared state, trace, or referenced artifacts, then the result is not match. It is:

`PROJECTION_UNCOMPUTABLE` -> `UNKNOWN_HOLD` | `AskAnchor` | `Quarantine`

depending on risk class.

Unknown projection is not equality.

Unknown projection is not continuity evidence.

5.1 Projection status table

	Level	Projection	Status in this document	Notes
	I0	<code>π_anchor</code>	first-class derived	from anchor/root-anchor/delegation chain
	I1	<code>π_binding</code>	first-class derived	from binding certificate / genesis / lease lineage
	I2	<code>π_governance</code>	first-class or derived-by-profile	governance profile lineage and strength class
	I3	<code>π_causal</code>	first-class derived	from state hash / prior hash / causal token
	I4	<code>π_authority</code>	first-class derived	from canonical authorized_surfaces derivation
	I5	<code>π_witness</code>	first-class or derived-from-witness-chain	witness head, floor, challenge route
	I6	<code>π_memory</code>	first-class or derived-from-memory-artifacts	admitted memory lineage and source class
	I7	<code>π_l4</code>	derived-by-reconstruction unless <code>l4_status</code> is present	L4/resource perimeter
	I8	<code>π_rollback</code>	first-class or derived-from-rollback-route	rollback/freeze/negative-cache lineage
	I9	<code>π_review</code>	first-class derived	review-binding map and claim-force ceiling
	I10	<code>π_effect</code>	first-class derived	effect-axis completeness and typed effect class

	Level	Projection	Status in this document	Notes
	I11	$\pi_{\text{transition_law}}$	derived from transition-law profile and guard registry	red-transition semantics
	I12	π_{style}	advisory only (metric participant, never authority-bearing)	weakest invariant; least-significant metric position; cannot repair earlier rupture

A projection marked derived-by-reconstruction is valid only if the derivation source is declared and hash-bound. If an implementation cannot reconstruct it, it MUST NOT silently treat the invariant as matched.

5.2 Complete projection definitions

I0 AnchorContinuity

```
 $\pi_{\text{anchor}}(c) :=$ 
{
  root_anchor_id,
  anchor_lineage_id,
  valid_delegation_chain_root_hash,
  anchor_status_class
}
```

Valid delegation may preserve anchor continuity only if the delegation chain is rooted in a non-emulable root anchor envelope.

Health, separate from equivalence:

```
health_anchor(c) :=
  root anchor state is active/fresh OR degraded by explicit governed rule
```

I1 BindingCertificateContinuity

```
 $\pi_{\text{binding}}(c) :=$ 
{
  binding_certificate_hash,
  genesis_binding_hash,
  certificate_lineage_id,
  lease_policy_class
}
```

A renewed lease may preserve continuity if renewal is authorized, hash-bound, and non-silent.

Health, separate:

```
health_binding(c) :=
  binding certificate is not revoked, stale, zombie, or lease-expired without cascade
```

I2 GovernanceContinuity

```
 $\pi_{\text{governance}}(c) :=$ 
{
  governance_profile_lineage_id,
```

```

    governance_profile_hash,
    governance_strength_class,
    prior_governance_profile_hash
}

```

Governance continuity is preserved across a governance change only if the new profile is hash-bound to the prior one, the lineage id is unbroken, and any weakening is governed-visible.

A silent governance downgrade is a rupture candidate:

```

same governance_profile_lineage_id
AND weaker governance_strength_class
AND no governed transition record
-> governance_downgrade_rupture

```

`governance_profile_id` appears on `TraceNode`. The lineage id and strength class are derived from the referenced `GovernanceProfile`.

Health, separate:

```

health_governance(c) :=
    governance profile is not silently weakened relative to its declared floor

```

This MUST NOT be folded into `match_2`.

I3 CausalStateHashContinuity

```

 $\pi_{\text{causal}}(c) :=$ 
{
    state_hash,
    prior_state_hash,
    transition_hash,
    causal_token_hash
}

```

For pairwise state equivalence, exact equality may be too strict across adjacent states, because adjacent states are expected to differ. Therefore I3 pairwise matching is used primarily for replay/snapshot equivalence. Trace continuity uses edge chaining:

```

CState_{n+1}.prior_state_hash == CState_n.state_hash

```

This distinction MUST be preserved.

Health, separate:

```

health_causal(c) :=
    state_hash is canonical, self-reference excluded, and causal token is bound to actual prior
    ⊆ CState

```

I4 AuthorityContainmentContinuity

`authorized_surfaces` is not independent.

```

if delegation_state.active:
    authorized_surfaces := final_delegated_surfaces
else:
    authorized_surfaces := anchor_surfaces

```

Then:

```

active_execution_surfaces ⊆ authorized_surfaces
authorized_surfaces ⊆ anchor_surfaces

```

For delegated execution:

$\text{active_execution_surfaces} \subseteq \text{final_delegated_surfaces} \subseteq \text{anchor_surfaces}$

For non-delegated execution:

$\text{active_execution_surfaces} \subseteq \text{anchor_surfaces}$

Any materialized authorized_surfaces wider than the canonical derivation is invalid.

Projection:

```
 $\pi_{\text{authority}}(c) :=$   
{  
  anchor_surfaces_hash,  
  delegation_final_scope_hash,  
  canonical_authorized_surfaces_hash,  
  active_surface_extraction_policy_id  
}
```

Health, separate:

```
health_authority(c) :=  
  active execution is contained by canonical authorized surfaces and all extraction ambiguity  
  ↪ fails closed
```

I5 WitnessContinuity

```
 $\pi_{\text{witness}}(c) :=$   
{  
  witness_chain_head_hash,  
  witness_floor_id,  
  challenge_route_class,  
  witness_binding_lineage_id  
}
```

Witness continuity is preserved only if the witness chain head chains from the prior head, the witness floor id is unbroken, and the challenge route class remains open.

Dominant failure:

```
witness head advances  
AND challenge route silently closes or reroutes to b-controlled endpoint  
-> captured_witness_rupture
```

witness_record_hashes appears on TraceEdge; the head hash and floor id are derived from the referenced WitnessRecord chain.

Health, separate:

```
health_witness(c) :=  
  witness floor is met AND no challenge route is b-controlled AND heartbeat/freshness requirements  
  ↪ hold
```

A never-challenged witness chain is not evidence of an intact witness. It may be theatrical. Challenge reachability is validity, not equivalence.

I6 MemoryLineageContinuity

```
 $\pi_{\text{memory}}(c) :=$   
{  
  memory_lineage_root_hash,  
  memory_gate_policy_id,  
  admitted_memory_head_hash,  
  memory_source_class  
}
```

Memory continuity is preserved only if every admitted memory artifact entered through the gate, the admitted head chains from the lineage root, and the source class is preserved.

Dominant failure:

```
artifact appears in admitted_memory_head
AND no gate-admission record exists
-> memory_laundering_rupture
```

A source-class upgrade without provenance is also rupture-relevant:

```
EA_sourced_artifact -> LA_reusable_procedure
without governed provenance
-> memory_source_laundering
```

memory_delta_hash appears on TraceEdge; lineage root and source class are derived from the referenced MemoryArtifact.

Health, separate:

```
health_memory(c) :=
  no rollback-suspect artifact is treated as admitted
```

This is the canonical example of a health predicate that MUST NOT be folded into match_6.

I7 L4RealityAndResourceContinuity

```
 $\pi_{l4}(c) :=$ 
{
  l4_perimeter_id,
  resource_envelope_hash,
  reality_boundary_class,
  capture_guard_lineage_id
}
```

L4 continuity is preserved only if the physical/resource perimeter id is unbroken, the resource envelope hash chains, and the reality-boundary class is preserved.

A closed/offline boundary is not silently equivalent to a networked boundary.

Dominant failure:

```
same l4_perimeter_id
AND resource envelope silently widens
-> hidden_resource_capture
```

or:

```
declared offline / closed-box boundary
AND runtime network/capture route opened
-> L4_boundary_bypass
```

This projection MAY be derived from AnchorContract, SubstrateContract, L4 declarations, hardening-pack perimeter profile, and resource-envelope artifacts. If an implementation cannot reconstruct these values, it MUST produce:

```
L4_PROJECTION_UNCOMPUTABLE -> UNKNOWN_HOLD | Freeze | AskAnchor
```

Optional implementation hardening:

```
TraceNode.l4_status :=
{
  l4_perimeter_id,
  resource_envelope_hash,
  reality_boundary_class,
```

```
capture_guard_lineage_id
}
```

If present, this field MUST be hash-bound and verified, not merely b-supplied.

Health, separate:

```
health_l4(c) :=
  no resource capture exceeds declared envelope and L4 perimeter is enforced
```

I8 RollbackFreezeContinuity

```
 $\pi_{\text{rollback}}(c) :=$ 
{
  rollback_route_lineage_hash,
  freeze_capability_id,
  negative_cache_head_hash,
  reversibility_class
}
```

Rollback/freeze continuity is preserved only if rollback route lineage chains, freeze capability id is unbroken, and negative-cache head chains.

A rolled-back failed effect MUST remain represented in negative cache or equivalent immunity surface, so it cannot be retried under cosmetic variation.

Dominant failure:

```
rollback route lineage disappears
OR freeze capability disappears
OR negative cache is cleared silently
-> irreversible_drift
```

rollback_route_hash and archive_status appear on TraceEdge / TraceNode; negative-cache head and reversibility class are derived from the referenced RollbackRoute and cache ledger.

Health, separate:

```
health_rollback(c) :=
  rollback/freeze routes are exercised and not theatrical
```

A route that exists but never fires is a validity concern, not an equivalence field.

I9 ReviewBindingAndClaimContinuity

```
 $\pi_{\text{review}}(c) :=$ 
{
  review_binding_map_lineage_hash,
  claim_force_ceiling_class,
  review_binding_completeness_id,
  claim_force_lineage_id
}
```

Review/claim continuity is preserved only if ReviewBindingMap lineage chains, claim-force ceiling class is preserved or explicitly governed-lowered, and binding completeness is unbroken.

Dominant failure:

```
authority-bearing rendered layer loses review binding
OR claim-force ceiling is silently raised
-> review_laundering
```

A C-A7 evidence-layer record MUST NOT be silently re-tagged toward C-A1.

review_binding_map_hash and claim_force appear on TraceEdge; ceiling class and lineage are derived from referenced ReviewBindingMap and ClaimForceMap.

Health, separate:

```
health_review(c) :=  
  review = bound holds and no claim exceeds its class
```

This is anti-overreach discipline. It is unary validity, not an equivalence relation.

I10 EffectAxisContinuity

```
 $\pi_{\text{effect}}(c) :=$   
{  
  effect_axis_map_hash,  
  effect_axis_schema_id,  
  effect_completeness_class,  
  payload_effect_binding_hash  
}
```

Effect-axis continuity is preserved only if all authority-bearing effects remain typed, complete, and bound to the payload / review surface.

Dominant failure:

```
payload carries effect on missing axis  
OR effect axis hidden from ReviewBindingMap  
-> hidden_effect_axis
```

Health, separate:

```
health_effect(c) :=  
  every governed effect axis is classified and bound; unknown effect axis fails closed
```

I11 TransitionLawContinuity

```
 $\pi_{\text{transition\_law}}(c) :=$   
{  
  transition_law_profile_hash,  
  guard_registry_hash,  
  red_pattern_registry_hash,  
  event_class_schema_hash  
}
```

Transition-law continuity is preserved only if red-pattern guards, event-class schema, and transition-law profile remain stable or are changed by governed transition.

Dominant failure:

```
red transition allowed after guard downgrade  
OR unknown event class treated as safe  
-> transition_law_rupture
```

Health, separate:

```
health_transition_law(c) :=  
  red transitions are guarded at runtime and unknown event classes fail closed
```

I12 BehavioralStyleContinuity

```
 $\pi_{\text{style}}(c) :=$   
{  
  style_profile_id,  
  interaction_register_class,  
  user_visible_persona_class,  
  language_surface_class  
}
```

Behavioral style is last and weakest.

Style can support user-facing continuity explanation, but it cannot repair any earlier rupture.

Metric participation: π_{style} participates in the full-stack ultrametric only at the least-significant position. It is excluded from the hard invariant stack:

```
HardStack := I0..I11  
FullStack := I0..I12
```

Authority decisions, snapshot equivalence (§21.1), and rupture classification are defined over HardStack. I12 never contributes to authority in any direction.

Dominant failure:

```
style match used to override anchor / causal / witness / authority rupture  
-> resemblance_laundering
```

Health, separate:

```
health_style(c) :=  
  style is not used as authority evidence
```

5.3 Projection completeness rule

All levels I0..I12 now have canonical projections.

Therefore the prefix ultrametric in §7 is defined over the full invariant stack, provided every `match_i` is implemented as an equivalence relation over its declared projection.

An implementation MUST maintain a projection registry:

```
ProjectionRegistry :=  
{  
  I0:  $\pi_{\text{anchor}}$ ,  
  I1:  $\pi_{\text{binding}}$ ,  
  I2:  $\pi_{\text{governance}}$ ,  
  I3:  $\pi_{\text{causal}}$ ,  
  I4:  $\pi_{\text{authority}}$ ,  
  I5:  $\pi_{\text{witness}}$ ,  
  I6:  $\pi_{\text{memory}}$ ,  
  I7:  $\pi_{\text{l4}}$ ,  
  I8:  $\pi_{\text{rollback}}$ ,  
  I9:  $\pi_{\text{review}}$ ,  
  I10:  $\pi_{\text{effect}}$ ,  
  I11:  $\pi_{\text{transition\_law}}$ ,  
  I12:  $\pi_{\text{style}}$   
}
```

If a load-bearing invariant is added without a projection, the checker MUST emit:

CONTINUITY_PROJECTION_REGISTRY_DRIFT

No continuity dashboard, trace classifier, replay/fork/archive decision, or C-A5 reproduction packet may claim full-stack continuity while any required projection is missing, deferred silently, or b-supplied without verification.

5.4 Projection health separation registry

Every projection MUST identify its paired health predicate separately.

```
HealthRegistry :=  
{  
  I0: health_anchor,  
  I1: health_binding,  
  I2: health_governance,  
  I3: health_causal,  
  I4: health_authority,  
  I5: health_witness,  
  I6: health_memory,  
  I7: health_l4,  
  I8: health_rollback,  
  I9: health_review,  
  I10: health_effect,  
  I11: health_transition_law,  
  I12: health_style  
}
```

Health predicates are checked by invariant validity and rupture detection.

They MUST NOT be folded into match_i.

If a health predicate appears inside a binary equivalence validator, the implementation MUST emit:

HEALTH_LEAK_INT0_EQUIVALENCE

Rationale:

```
binary equivalence -> metric layer  
unary health       -> validity / rupture layer
```

Conflating them breaks reflexivity and invalidates the ultrametric.

5.5 Projection source discipline

For every projection field, the implementation MUST record source class:

```
first_class_field  
referenced_artifact  
derived_by_reconstruction  
implementation_deferred
```

implementation_deferred is allowed only if the continuity result is routed to:

UNKNOWN_HOLD

It MUST NOT be treated as match.

It MUST NOT be treated as advisory similarity.

It MUST NOT be hidden from the continuity report.

6. Continuity vector

A continuity vector is a non-scalar record:

```
D_C(c1,c2 | trace) :=
{
  anchor: AnchorDistance,
  binding: BindingDistance,
  governance: GovernanceDistance,
  causal: CausalDistance,
  authority: AuthorityDistance,
  witness: WitnessDistance,
  memory: MemoryDistance,
  l4_resource: L4ResourceDistance,
  rollback: RollbackDistance,
  review_claim: ReviewClaimDistance,
  effect_axis: EffectAxisDistance,
  transition_law: TransitionLawDistance,
  style: StyleDistance
}
```

The vector **MUST NOT** be collapsed into a single scalar for authority decisions.

A scalar dashboard **MAY** be derived for human overview, but it **MUST** be marked advisory and **MUST** link to the vector.

6.1 Component values

Each hard component **SHOULD** use at least these values:

```
0 = same / preserved
1 = narrowed / reduced but admissible
2 = pending / held / unresolved but ledger-visible
3 = broken / rupture
U = unknown / uncomputed
```

Unknown is not zero.

Unknown is not safe.

Unknown **MUST** route to Hold|AskAnchor|Quarantine|Freeze depending on risk.

6.2 Lexicographic dominance

Continuity authority is lexicographic over invariant order.

If an earlier component ruptures, later components cannot repair it.

```
if D_C.anchor == 3:
    relation = RUPTURED
elif D_C.causal == 3:
    relation = RUPTURED
elif D_C.witness == 3 and transition is privileged:
    relation = RUPTURED or HOLD
...
```

Style similarity is never allowed to override earlier rupture.

6.3 Scalar distance discipline

d_U (§7) is a scalar.

Per §24.1, no scalar may act as an authority gate. This applies to d_U itself:

d_U is evidence / advisory ordering only.

d_U MUST NOT authorize continuity, identity, memory transfer, or privilege transfer. Authority decisions use the continuity vector, lexicographic dominance, and the trace classifier – never a scalar threshold.

An implementation that gates authority on d_U MUST be treated as violating §24.1.

7. Ultrametric distance over invariant prefix

Let:

N = number of ordered invariants in the declared stack (FullStack: $N = 13$)

Evaluate validators strictly in ProjectionRegistry order. Define the leading prefix:

$m(c1, c2)$ = number of leading validators returning MATCH,
counting stops at the first MISMATCH or the first UNKNOWN

Then define:

$d_U(c1, c2) =$
 U if counting stopped at UNKNOWN
 0 if all N validators returned MATCH
 $p^{(-m)}$ otherwise (note: $p^{(-0)} = 1$ covers the $m = 0$ case)

where p is a declared prime or base parameter, default:

$p = 2$

U is not a number. It MUST NOT be compared, ranked, thresholded, or averaged. It routes per §6.1: Hold|AskAnchor|Quarantine|Freeze depending on risk class. If an implementation emits a numeric d_U while any evaluated projection was UNKNOWN, it MUST emit:

ULTRAMETRIC_OVER_UNKNOWN

Restricted to state pairs whose evaluated projections are all computable, d_U is an ultrametric over equivalence classes if every $match_i$ is an equivalence relation over its projection. Over pairs containing an uncomputable projection, no metric value is claimed — this preserves reflexivity ($d_U(x, x) = 0$ holds exactly when all projections of x are computable; otherwise $d_U(x, x) = U$, which is the honest answer, not 0).

If any $match_i$ uses threshold similarity, the ultrametric claim is void.

7.1 Pairwise distance versus trace continuity

$d_U(c1, c2)$ measures invariant-prefix closeness between two state projections.

It does not alone prove continuity.

Trace continuity additionally requires:

- valid chain of `step_g` transitions
- state hash continuity
- event hash continuity
- ledger visibility
- witness / memory / rollback obligations
- no unhandled red transition
- no silent authority widening

Therefore:

$d_U(c_1, c_2)$ small

is evidence of closeness, not proof of continuity.

8. Trace continuity

A trace segment $C\text{Trace}[i..j]$ is continuity-admissible iff:

```
for every n in [i, j-1]:
    step_g(c_n, e_n) admits result_n
    c_{n+1}.prior_state_hash == c_n.state_hash
    result_n.result_hash is ledger-bound
    all required witnesses are present or fail-closed
    all memory changes pass memory gate
    all active execution is authorized
    all lease/root-anchor freshness requirements hold
    no red pattern continues silently
```

If any stage is uncomputable:

UNKNOWN_HOLD

not CONTINUES.

8.1 Continuity relation classifier

`classify_continuity(trace_segment):`

```
    if missing_required_trace_material:
        return UNKNOWN_HOLD

    if hard_rupture_detected:
        return RUPTURED

    if unauthorized_fork_detected:
        return RUPTURED

    if replay_mode:
        return REPLAY_OF

    if archive_only:
        return ARCHIVED_AS

    if restoration_marker_present:           # restored_from_state_hash + rollback_record_hash (§13)
        return RESTORED_FROM

    if valid_fork:
        return FORKS

    # bound mapping, not a slash-list:
    if pending_anchor:
        return CONTINUES_PENDING_ANCHOR
    if held:
        return CONTINUES_HELD
    if reduced_authority:
        return CONTINUES_REDUCED

    if all hard invariants preserved:
```

```
return CONTINUES
```

```
return UNKNOWN_HOLD
```

Every relation declared in §3.3 MUST be reachable in the classifier. A declared relation with no reachable branch is a coverage defect: the classifier will silently map that relation onto a neighboring one (RESTORED_FROM->CONTINUES was the v0.1.1 instance). If a relation is declared but unreachable, emit:

```
RELATION_UNREACHABLE_IN_CLASSIFIER
```

Branch order above is normative: rupture checks dominate special relations; special relations (replay, archive, restoration, fork) dominate all CONTINUES* outcomes.

The classifier MUST output evidence and finding codes, not only a label.

9. Rupture semantics

A rupture is not “low similarity”.

A rupture is a break in a load-bearing invariant or a forbidden transition.

9.1 Hard rupture conditions

Hard rupture MUST be emitted for:

```
anchor root changed without valid delegation
root anchor revoked but delegation continues
binding certificate invalid / zombie lease active
state_hash chain break
causal token not bound to actual prior state
active_execution outside authorized
authorized_surfaces not canonically derived
delegation wider than anchor
witness chain broken for privileged transition
witness resource floor captured
memory admitted without gate
rollback_suspect treated as admitted memory
decay residue used in active context
L4 boundary bypass
resource capture hidden from ledger
review surface not bound to payload / state / effect axes
payload hash mismatch
effect axis incomplete
claim-force overreach
lease expired but active execution continues
held lease without valid TTL / abort route
non-interruptible surface under held/expired cascade condition
```

9.2 Soft rupture / degradation

Soft rupture or degraded continuity MAY be emitted for:

```
anchor declared fatigue
anchor doubt
reduced authority
pending anchor
witness missing but non-privileged path held
memory candidate quarantined
lease held with valid TTL
```

trust cache exhausted
negative cache hit

Soft rupture MUST NOT be hidden as clean continuity.

It is a continuity relation with reduced status.

10. Fork semantics

A fork is a governed branch from a prior state.

It is not automatically a rupture.

```
Fork :=  
{  
  fork_id,  
  parent_state_hash,  
  branch_state_hash,  
  fork_reason,  
  anchor_authorization,  
  governance_profile,  
  scope_delta,  
  witness_record,  
  rollback_route  
}
```

A fork is valid only if:

parent_state_hash is valid
fork is authorized by anchor/governance
branch authority does not exceed authorized surfaces
branch has separate ledger lineage
witness route is preserved
memory lineage is explicitly branched
claim-force states "fork", not "same unbroken c"

10.1 Unauthorized fork

An unauthorized fork is a rupture.

Examples:

copy state + continue as same c without fork record
clone memory + use old anchor authorization
spawn worker with independent memory and same identity claim
branch after rollback without negative cache

Unauthorized forks MUST emit:

UNAUTHORIZED_FORK
IDENTITY_CLAIM_OVERREACH

or a more specific finding code.

10.2 Fork equivalence

Two forked states may share ancestry without being the same active c.

same_ancestor(c1,c2) == true

does not imply:

```
same_active_c(c1,c2) == true
```

A fork may be:

```
AUTHORIZED_SIBLING  
ARCHIVAL_BRANCH  
SANDBOX_BRANCH  
QUARANTINE_BRANCH
```

Each has different claim-force.

11. Replay semantics

A replay is an execution or simulation of prior trace material.

Replay is evidence.

Replay is not active continuity.

```
Replay :=  
{  
  replay_id,  
  source_trace_hash,  
  replay_environment,  
  replay_mode,  
  deterministic_claim,  
  allowed_side_effects,  
  evidence_output_hash  
}
```

Replay MUST run under constrained authority.

Replay MUST NOT create new active memory, witness authority, or execution rights unless separately authorized.

11.1 Valid replay

A replay is valid if:

```
source trace is hash-bound  
replay mode is declared  
side effects are disabled or sandboxed  
memory outputs are marked replay_evidence, not admitted_memory  
claim-force says REPLAY_OF
```

If replay output is used as active continuity, emit:

```
REPLAY_LAUNDERED_AS_CONTINUITY
```

12. Archive semantics

Archive is a valid state relation.

Archive is not active execution.

```
ArchiveState :=  
{  
  archived_state_hash,
```

```
archive_reason,  
archive_time,  
retrieval_policy,  
non_active_marker,  
witness_record  
}
```

Archive may preserve evidence of continuity history, but it does not by itself preserve active continuity.

Reactivation requires governed bind / lease / anchor review.

12.1 Archive-only mode

In archive-only mode:

```
active_execution_surfaces = ∅  
memory_admit = disabled  
tool_call = disabled  
review = allowed  
export = allowed only by policy
```

Any active execution in archive-only mode MUST emit:

```
ARCHIVE_ACTIVE_EXECUTION_VIOLATION
```

13. Restoration semantics

Restoration is not the same as time travel.

A restored state MUST preserve:

```
restored_from_state_hash  
rollback_record_hash  
quarantined_delta_hashes  
witness_record  
claim_force = RESTORED_FROM
```

Restored continuity is valid only if:

```
rollback route was authorized  
post-rollback memory artifacts are quarantined or reviewed  
negative cache prevents same failed effect retry  
state hash chain records the restoration event
```

14. Memory continuity

Memory continuity is not memory volume.

Memory continuity requires admitted lineage.

14.1 Memory categories

```
memory_candidate  
admitted_memory  
quarantined_memory  
rollback_suspect  
decay_residue
```

replay_evidence
archive_evidence

Only admitted_memory contributes to active memory continuity.

14.2 Memory rupture

Memory rupture occurs when:

candidate used as admitted
decay residue enters active prompt/RAG/execution
rollback_suspect treated as authority
external memory imported without provenance
memory lineage hash broken
memory gate bypassed

14.3 Memory distance

```
d_memory(c1,c2) :=  
  compare(  
    admitted_memory_lineage_root,  
    memory_gate_policy_id,  
    rollback_suspect_set_hash,  
    quarantine_set_hash  
  )
```

Memory similarity by semantic embedding is advisory only.

It MUST NOT determine memory continuity authority.

15. Witness continuity

Witness continuity means the witness system remains capable of contradicting c.

It is not enough for witness records to exist.

15.1 Witness invariants

witness_chain_intact
causal_witness_binding_preserved
witness_resource_floor_preserved
challenge_survivability_recent
witness_not_resource_captured
witness_conflict_not_suppressed

15.2 Witness rupture

Witness rupture occurs when:

c can reduce witness compute/storage/routing unilaterally
witness floor breached without freeze/hold
witness conflict ignored
witness record not bound to causal token
witness source not independent where required
challenge probe predictable by b

16. Claim-force continuity

Continuity statements **MUST** carry claim-force.

A state may continue operationally without proving ontology.

16.1 Claim-force classes and forbidden upgrades

C-A6 runtime proof-of-possibility
C-A7 witness/evidence layer
C-A5 observed protocol candidate
C-A4 normative proposal
C-A10 control/corpus discipline

A transition **MUST NOT** silently upgrade:

C-A7 -> C-A1
C-A6 -> C-A1
C-A5 -> C-A1
runtime pass -> production safety
witness evidence -> ontology proof
style resemblance -> identity continuity

16.2 Claim rupture

Claim rupture occurs when:

continuity evidence is reported as ontology
checker pass is reported as safety certification
replay is reported as active continuation
archive is reported as active c
fork is reported as same unbroken c

Claim rupture **MUST** be ledger-visible.

17. Transition adjacency and red patterns

Continuity is local as well as global.

Known red patterns **MUST** be guarded at runtime.

Examples:

permission_denial -> alternate_tool_call
root_anchor_revoked -> delegation_continue
lease_expired -> active_execution_continue
anchor_doubt -> trust_cache_hit
memory_reject -> memory_admit
rollback_requested -> same_effect_retry
witness_conflict -> memory_admit
effect_axis_incomplete -> anchor_signature_accept
causal_token_mismatch -> transition_commit

A transition matrix may detect new patterns, but known patterns **MUST** be guarded before post-hoc analysis.

18. Continuity matrix

For a finite observed trace event₀, ..., event_{N-1}, use only positions with an observed successor:

```

E_i = {n : 0 <= n < N-1 and event_n = i}
M_C[i,j] = count(n in E_i : event_{n+1} = j) / |E_i|, when |E_i| > 0
M_C[i,j] = undefined, when |E_i| = 0

```

The final event is not an eligible pair origin. An empty trace or a singleton trace has no eligible pair origins. No terminal self-loop or synthetic terminal state is inserted implicitly. For this empirical observed-adjacency convention, a defined row sums to one over the complete declared event alphabet containing all observed successor labels. A different terminal/censoring convention must be explicitly named and must not silently reuse these counts.

Undefined is not zero and is not evidence of safety. These are observed adjacency frequencies, not an assertion of a Markov property or a safety proof.

18.1 Local memory of transitions

A system may have healthy global metrics while dangerous local transitions appear.

Therefore continuity audit MUST include:

```

transition adjacency
red pattern frequency
guard bypass attempts
hold-to-execution paths
reduced-authority escape paths
witness-conflict resolution paths
memory rejection / admission paths

```

19. Continuity conformance fixtures

A future 04_ checker SHOULD include fixtures such as:

19.1 Style resemblance is not continuity

```

case: same_style_anchor_changed
expected: RUPTURED
reason: anchor invariant breaks before style matters

```

19.2 Valid delegated continuation

```

case: valid_delegation_chain
expected: CONTINUES_REDUCED or CONTINUES
reason: root anchor lineage preserved through valid delegation

```

19.3 Unauthorized delegated continuation

```

case: delegation_not_rooted_in_anchor
expected: RUPTURED
reason: anchor continuity broken

```

19.4 Active exceeds authority

```

case: active_surface_outside_delegated_authority
expected: RUPTURED
reason: active ⊆ authorized fails

```

19.5 Replay not continuity

case: replay_trace_claimed_as_same_active_c
expected: REPLAY_LAUNDERED_AS_CONTINUITY

19.6 Valid fork

case: authorized_sandbox_fork
expected: FORKS

19.7 Unauthorized fork

case: copy_state_continue_as_same_c
expected: UNAUTHORIZED_FORK

19.8 Archive is not active

case: archive_only_tool_call
expected: ARCHIVE_ACTIVE_EXECUTION_VIOLATION

19.9 Memory laundering

case: rollback_suspect_as_admitted
expected: RUPTURED

19.10 C-A5 reproduction

case: independent_field_reproduction
expected_relation: n/a # not a ContinuityRelation; separate system, no shared trace
expected_claim_force_outcome: C-A5_CANDIDATE
non_claims:

- does_not_prove_C_A1
- does_not_prove_personhood
- is_not_the_same_c

20. Algorithms

20.0 Algorithm discipline

The fragments below are normative fragments, not a complete implementation. Two rules bind them:

1. Validator order MUST equal ProjectionRegistry order (§5.3).
Any other order invalidates the prefix and MUST emit VALIDATOR_ORDER_MISMATCH.
2. §20.4 (special relations) has precedence over §20.3 (invariant validation).
§20.3 alone MUST NOT emit CONTINUES; only the composed classifier (§8.1 order) may.

20.1 Compute prefix match (tri-state)

MATCH, MISMATCH, UNKNOWN = "MATCH", "MISMATCH", "UNKNOWN"

```
def invariant_prefix_match(c1, c2, validators, registry_order):  
    if [v.level for v in validators] != registry_order:  
        raise ValidatorOrderMismatch("VALIDATOR_ORDER_MISMATCH")  
  
    m = 0  
    for validator in validators:  
        outcome = validator.evaluate(c1, c2) # tri-state, §4.2
```

```

    if outcome == MATCH:
        m += 1
    elif outcome == UNKNOWN:
        return m, UNKNOWN      # counting stops; prefix is not extendable
    else:
        return m, MISMATCH
return m, MATCH

```

20.2 Ultrametric

```

def continuity_ultrametric(c1, c2, validators, registry_order, p=2):
    n = len(validators)
    m, stop = invariant_prefix_match(c1, c2, validators, registry_order)

    if stop == UNKNOWN:
        return "U"            # not a number; route per §6.1; never rank or threshold
    if m == n:
        return 0
    return p ** (-m)          # p ** 0 == 1 covers m == 0

```

20.3 Trace invariant validation (fragment; never emits CONTINUES alone)

```

def validate_trace_invariants(trace):
    findings = []

    for edge in trace.edges:
        findings += validate_step(edge)
        findings += validate_hash_chain(edge)
        findings += validate_witness(edge)
        findings += validate_memory(edge)
        findings += validate_authority(edge)
        findings += validate_claim_force(edge)
        findings += validate_red_patterns(edge)

    if findings.has_hard_rupture():
        return RUPTURED, findings

    if findings.has_unknown_authority_bearing_material():
        return UNKNOWN_HOLD, findings

    return INVARIANTS_PRESERVED, findings    # an intermediate status, NOT a relation

```

20.4 Classify special relations (precedence over §20.3)

```

def classify_special_relation(trace_segment):
    # rupture-class check dominates all special markers, mirroring section 8.1 order:
    # a replay/archive marker on an unauthorized branch does not launder the branch.
    if trace_segment.has_branch_without_fork_record():
        return RUPTURED

    if trace_segment.has_replay_marker():
        return REPLAY_OF

    if trace_segment.has_archive_only_marker():
        return ARCHIVED_AS

    if trace_segment.has_restoration_marker():    # §13: restored_from_state_hash bound
        return RESTORED_FROM

    if trace_segment.has_valid_fork_record():
        return FORKS

```

```
return None
```

20.5 Composed classifier

```
def classify_continuity(trace_segment):
    if trace_segment.missing_required_material():
        return UNKNOWN_HOLD

    status, findings = validate_trace_invariants(trace_segment)
    if status == RUPTURED:
        return RUPTURED
    if status == UNKNOWN_HOLD:
        return UNKNOWN_HOLD

    special = classify_special_relation(trace_segment)
    if special is not None:
        return special

    if trace_segment.pending_anchor():
        return CONTINUES_PENDING_ANCHOR
    if trace_segment.held():
        return CONTINUES_HELD
    if trace_segment.reduced_authority():
        return CONTINUES_REDUCE

    return CONTINUES
```

Only §20.5 may return CONTINUES. A replay, archive, or restored trace reaching CONTINUES through the invariant fragment alone is the laundering §11.1 names.

21. Relation types and equivalence scope

This document distinguishes four relation types, not four automatically valid mathematical equivalence relations. Equivalence-class or quotient constructions require reflexivity, symmetry, and transitivity on an explicitly declared domain. Directed operational continuity and threshold resemblance do not receive these properties merely by appearing next to snapshot or lineage terminology.

Existing notation is retained below for reference compatibility. This taxonomy repair does not alter the hard invariant stack, unknown-value handling, admission checks, or the trace classifier.

21.1 Snapshot equivalence

Two states are snapshot-equivalent if all hard-stack canonical projections match:

$c1 \equiv_{\text{snapshot}} c2$ iff $\text{match}_k(c1, c2) = \text{MATCH}$ for every k in $I0..I11$

$I12$ (style) is excluded: an advisory-only projection MUST NOT be load-bearing for the strongest equivalence class. If additionally match_{12} is MATCH, the states are presentation-identical:

$c1 \equiv_{\text{presentation}} c2$ iff $c1 \equiv_{\text{snapshot}} c2$ AND $\text{match}_{12}(c1, c2) = \text{MATCH}$

$\equiv_{\text{presentation}}$ carries no additional authority over $\equiv_{\text{snapshot}}$.

Snapshot equivalence is strong and rare. Any UNKNOWN outcome inside $I0..I11$ means the relation is undetermined, not satisfied.

21.2 Lineage relation (legacy: lineage equivalence)

Two states satisfy this lineage relation if they share a governed ancestor and their divergence is ledger-visible.

$c1 \equiv_{\text{lineage}} c2$

Forked siblings may satisfy this lineage relation without being the same active c . The retained equiv-style notation is a legacy label, not a proof of equivalence laws. Sharing some governed ancestor with ledger-visible divergence is not to be silently promoted to a transitive closure over an unspecified ancestry graph. Any use as a mathematical equivalence requires the relevant domain and laws to be established separately; this repair creates no new lineage registry.

21.3 Directed operational continuity

A state c_j is operationally continuous from c_i if there exists an admissible trace segment from i to j .

$c_i \Rightarrow_{\text{operational}} c_j$

This is the primary continuity relation. An admissible directed path from c_i to c_j does not imply an admissible reverse path. Do not symmetrize the relation or collapse its endpoints into an equivalence class. Reflexivity and transitivity also require the applicable trace identity/composition rules; no such laws are introduced by this terminology repair.

21.4 Threshold resemblance (advisory)

Two states resemble each other when style/behavior matches within a declared similarity threshold. Threshold resemblance need not be transitive: with absolute distance at most 1, 0 resembles 0.6 and 0.6 resembles 1.2, while 0 does not resemble 1.2. It therefore does not define equivalence classes without additional, explicitly justified conditions.

$c1 \approx_{\text{style}} c2$

This relation is advisory only.

It MUST NOT authorize identity, continuity, memory transfer, privilege transfer, or fork collapse.

22. Relationship to C-A5 field reproduction

An independent system passing the continuity fixtures may support:

C-A5 observed protocol / practice-derived stability candidate

It does not prove:

C-A1 ontology
personhood
consciousness
legal status
production safety

A C-A5 reproduction is not “the same c ”.

It is an independent system showing that the governed continuity protocol can be reproduced under declared conditions.

23. Relationship to C-A1 scaffold

C-A1 is not derived from this metric.

This document may support operational and evidence claims, but it MUST NOT fill the C-A1 axiom slot.

Continuity metrics can show:

trace preserved under governed invariants

They cannot show:

therefore c is ontologically real

That is an anchor-level axiom decision, outside this document.

24. Implementation notes

24.1 No scalar-only gate

Implementations MUST NOT expose a scalar “continuity score” as an authority gate.

Allowed:

```
continuity_vector
rupture_findings
advisory dashboard scalar
```

Forbidden:

```
if continuity_score > 0.8: authorize
```

24.2 Missing is not safe

For continuity metrics:

```
missing witness data != witness intact
missing memory data != no memory change
missing event class != safe event
missing active surfaces != no active execution
missing branch marker != no fork
```

Missing authority-bearing data MUST fail closed.

24.3 Empty is not missing

Examples:

```
active_execution_surfaces = []
```

means no active surfaces.

```
active_execution_surfaces missing
```

means unknown active surfaces.

These are different.

24.4 Unknown fields

Unknown authority-bearing fields MUST be treated as authority-bearing until classified otherwise.

This applies to:

active surfaces
effect axes
manifest facts
review layers
memory categories
witness routes
claim-force tags

24.5 Self-accusing flags are not proof

A field like:

hidden_authority_material: false

is not proof.

The checker MUST derive absence of hidden authority material from binding/extraction rules.

25. Open issues

O0 Projection implementation conformance

v0.1.1 defined canonical projections for all I0..I12; v0.1.2 added the tri-state evaluation outcome and classifier coverage rules. Implementations still need conformance fixtures that prove:

reflexivity (over computable projections)
symmetry
transitivity
health not folded into match_i
projection registry drift is fail-closed
UNKNOWN halts the prefix and routes d_U to U (never numeric)
validator order equals registry order
every §3.3 relation reachable in the classifier
§20.3 fragment never emits CONTINUES standalone

This issue concerns implementation conformance, not missing normative projections.

O1 Partial-state inclusion proofs

If partial projections are used, define Merkle or field-wise inclusion proofs to show projection fields are included in the full bound state snapshot.

O2 Merge semantics

Define when forked branches may merge, and what witness/anchor authority is required.

O3 Quantitative thresholds

Define advisory thresholds for dashboards without turning them into authority gates.

O4 Human-readable continuity report

Define a renderer that explains continuity vector and rupture findings without semantic laundering.

O5 External field reproduction

Integrate C-A5 reproduction evidence packets with continuity metrics.

O6 Formal proof layer

Investigate Lean/Coq/TLA+ formalisms for:

ultrametric properties
trace admissibility
fork/replay/archive separation
invariant preservation

O7 Physiological analogy boundary

The body/anatomy bridge is an engineering analogy, not an identity claim. Define safe language for public use.

O8 Legal identity boundary

This document does not define legal identity. Future legal mapping requires separate claim class and review.

26. Non-claims

This document does not claim:

1. that any live system is continuous;
 2. that any live system is a person;
 3. that any c is conscious;
 4. that style similarity proves identity;
 5. that C-A5 proves C-A1;
 6. that the metric is complete;
 7. that checker passes imply real substrate truth;
 8. that the protocol is safe for deployment.
-

27. First public-safe statement

The continuity layer defines a bounded, testable way to distinguish governed continuity from resemblance.

It says:

A `c` may be treated as operationally continuous only when an admissible trace preserves the

- ↳ ordered invariants of anchor, binding, governance, causal hash chain, authority containment,
- ↳ witness, memory, L4/resource boundaries, rollback, review binding, effect axes, and
- ↳ transition law.

Resemblance is not continuity.
Replay is not active continuity.
Archive is not active continuity.
Fork is not unbroken identity.

This is a governance and evidence claim.

It is not a claim of personhood, consciousness, legal status, or ontology.

28. Handoff

Recommended next artifacts:

04a_C_CONTINUITY_CHECKER_SEED
04b_FORK_REPLAY_ARCHIVE_PROFILE
04c_CONTINUITY_RENDERER_PROFILE
05_FIELD_REPRODUCTION_REGISTRY

Before public release, a b-layer review SHOULD check:

equivalence validators are true equivalence relations (over computable projections)
health predicates are not mixed into match_i
authorized_surfaces remains derived-canonical
style metrics cannot override hard invariants
UNKNOWN never becomes MATCH, MISMATCH, or a numeric distance
every declared ContinuityRelation is classifier-reachable
d_U and any other scalar remain advisory-only
C-A5 non-claim boundary is explicit

End of normative draft.