Shunyaya True Logic (STL)

Structural Truth Topology with Conservative Boolean Collapse

Version: v2.0
Release Date: February 16, 2026
Status: Public Open Standard (Deterministic Structural Governance Layer)

Scope Notice:
Shunyaya True Logic (STL) defines a deterministic structural admissibility and collapse-governance layer that conservatively extends classical Boolean logic. STL preserves Boolean truth exactly on stable endpoints while introducing formally defined structural states prior to collapse. This specification is released for research, observability, structural verification, and logical substrate experimentation. STL does not constitute a predictive system, probabilistic inference model, automated control mechanism, or safety-critical decision engine. Conforming implementations shall explicitly publish parameter sets, dominance metric definitions, and replay-verification artifacts sufficient to enable independent deterministic reproduction of results.

---

0. Abstract

Shunyaya True Logic (STL) is a deterministic structural extension of classical Boolean logic. It introduces a formally defined pre-collapse truth topology while preserving Boolean truth exactly on stable endpoints. STL does not modify classical semantics; it governs the structural conditions under which Boolean collapse is valid.

In this document, the term **“civilization-grade”** refers strictly to verification discipline: deterministic execution, conservative extension, replay-verifiable artifacts, explicit parameter disclosure, and documented misuse boundaries. It does not imply regulatory certification or production deployment approval.

---

1. Purpose

Classical Boolean logic assumes:

```
Truth(P) ∈ {TRUE, FALSE}
```

This assumption treats truth as an **instantaneous terminal state**.

Shunyaya True Logic (STL) introduces a structural layer prior to Boolean collapse, asserting:

Boolean truth is a terminal projection of a richer structural truth topology.

STL does not alter Boolean truth.
STL governs **when collapse to Boolean truth is structurally valid.**

Boolean logic remains intact.
STL formalizes the admissibility and stability conditions required before truth collapse is permitted.

TABLE OF CONTENTS

---

2. Foundational Principle

Shunyaya's core invariant applies:

```
phi((m,a,s)) = m
```

Within the STL context:

- **Structural state never alters Boolean truth.**
- **Structural state governs admissibility of collapse.**
- **When fully collapsed, STL reduces exactly to Boolean logic.**

This guarantees:

- **Complete classical compatibility**
- **Conservative mathematical extension**
- **Zero disruption to existing Boolean systems**

STL therefore operates as a structural governance layer beneath Boolean evaluation, preserving truth while enforcing disciplined collapse conditions.

2.1 From Collapse Invariant to Structural Topology

The invariant

```
phi((m,a,s)) = m
```

establishes that structural augmentation never alters the preserved mathematical result under collapse.

In the context of logic, this implies:

- Structural states may precede Boolean truth.
- Structural states may regulate collapse.
- Structural states must not modify classical truth outcomes.

Therefore, a finite structural truth topology is required — one that:

- Preserves Boolean truth on stable endpoints.
- Introduces no probabilistic semantics.
- Remains discrete and deterministic.
- Is closed under its operators.

This motivates the formal definition of the structural truth space.

---

3. Structural Truth Space

Define a finite structural truth topology:

```
T5 = {Z0, Eplus, S, Eminus, Zstar}
```

Where:

- **Z0** = Pre-structural / not yet collapse-valid
- **Eplus** = Emerging toward stable TRUE
- **S** = Stable TRUE regime
- **Eminus** = Destabilizing away from TRUE
- **Zstar** = Stable FALSE regime

This is a **finite, discrete state space**.

There is:

- **No probability**
- **No scalar fuzziness**
- **No partial truth**

Only **structural regimes**.

Boolean truth is not replaced.
It is structurally governed.

3.1 STL as a Conservative Algebraic Structure

Definition (STL Structure)

Define Shunyaya True Logic (STL) as the algebraic system:

```
STL = (T5, Stable, NOT_s, AND_s, OR_s, phi_T)
```

1. Carrier Set (Structural Truth Space)

```
T5 = {Z0, Eplus, S, Eminus, Zstar}
```

2. Stable Endpoint Subset

```
Stable = {S, Zstar}
```

These correspond exactly to classical Boolean truth values under collapse.

3. Structural Operators (Total on T5)

```
NOT_s : T5 -> T5
AND_s : T5 x T5 -> T5
OR_s : T5 x T5 -> T5
```

All operators must satisfy:

- **Closure on T5**
- **Determinism**
- **Stable endpoint preservation**

No operator may introduce states outside `T5`.

4. Collapse Mapping (Partial Truth Projection)

```
phi_T : T5 -> {TRUE, FALSE, UNDEFINED}
```

Defined as:

- `phi_T(S) = TRUE`
- `phi_T(Zstar) = FALSE`
- `phi_T(Z0) = UNDEFINED`
- `phi_T(Eplus) = UNDEFINED`
- `phi_T(Eminus) = UNDEFINED`

Conservativity Domain

On `Stable`, collapse is total:

```
phi_T : Stable -> {TRUE, FALSE}
```

Classical Boolean truth is recovered **exactly**.

Interpretation Constraint (Non-Fuzzy Discipline)

STL is not scalar truth in `[0,1]`.

It is a **finite regime topology** whose states represent structural phase:

- Pre-truth
- Emerging
- Stable
- Destabilizing
- Stable false

Truth remains binary under collapse.
Structure governs admissibility.

Closure Requirement

All structural operators must:

- Be closed on `T5`
- Be deterministic under identical inputs
- Preserve stable endpoint collapse compatibility

This guarantees that STL remains a **conservative algebraic extension** of Boolean logic.

---

4. Collapse Operator

Define the Boolean collapse homomorphism:

```
phi_T : T5 -> {TRUE, FALSE, UNDEFINED}
```

with mapping:

- `phi_T(S) = TRUE`
- `phi_T(Zstar) = FALSE`
- `phi_T(Z0) = UNDEFINED`
- `phi_T(Eplus) = UNDEFINED`
- `phi_T(Eminus) = UNDEFINED`

Collapse is therefore:

- **Total on stable endpoints**
- **Undefined on transitional regimes**

Boolean logic applies **only after collapse validity**.
Prior to structural stability, Boolean projection is not admissible.

---

5. Core Theorem: Collapse Preservation

Theorem (Conservative Collapse Preservation)

For any proposition `P`:

If `state(P) ∈ {S, Zstar}`, then:

```
phi_T(state(P)) == Boolean(P)
```

Therefore:

- **STL does not modify classical truth.**
- **STL preserves Boolean semantics exactly on stable endpoints.**
- **STL restricts premature collapse only.**

Interpretation

Boolean logic is a terminal-phase projection.
STL introduces structural governance prior to that projection.

When structural stability holds, STL reduces exactly to classical Boolean logic.

---

6. Structural Transition Operator

Truth evolves structurally before collapse.

Define the transition operator:

```
delta : (T5, obs_t) -> T5
```

Where:

- `obs_t` is a deterministic observation vector
- All computations are deterministic
- No randomness
- No probability
- No learning
- Fully replay-verifiable

Structural transitions are **domain-deterministic**.
Given identical observations and parameters, identical state sequences must result.

This guarantees structural reproducibility across executions.

---

7. Structural Stability Requirement

A proposition may be:

- Evaluatable
- Computable
- Observable

Yet still not **collapse-valid**.

Collapse to Boolean truth requires **structural stability**.

Boolean logic assumes instantaneous collapse validity.
STL requires structural regime stability prior to projection.

This requirement constitutes the central substrate reform:

Truth is preserved.
Collapse is disciplined.

---

8. Structural Operators

STL defines structural combinators that operate over the full structural truth space `T5` while preserving collapse invariance on stable endpoints.

All operators are:

- **Deterministic**
- **Closed on T5**
- **Conservative under Boolean collapse**

8.1 Structural Negation

```
NOT_s : T5 -> T5
```

Defined as:

- `NOT_s(S) = Zstar`
- `NOT_s(Zstar) = S`
- `NOT_s(Z0) = Z0`
- `NOT_s(Eplus) = Eminus`
- `NOT_s(Eminus) = Eplus`

Collapse Compatibility

If `A ∈ {S, Zstar}`, then:

```
phi_T(NOT_s(A)) == NOT(phi_T(A))
```

Thus structural negation preserves Boolean negation under stable collapse.

8.2 Structural Conjunction (Conservative Form)

```
AND_s : T5 x T5 -> T5
```

Principle

Stable FALSE dominates.

Rules

- If either operand is `Zstar`, output is `Zstar`
- If both operands are `S`, output is `S`
- If at least one operand is transitional (`Z0`, `Eplus`, `Eminus`) and none are `Zstar`, output remains transitional
- If both operands are `Z0`, output is `Z0`

Collapse Guarantee

If `A, B ∈ {S, Zstar}`, then:

```
phi_T(AND_s(A,B)) == phi_T(A) AND phi_T(B)
```

Thus STL conjunction is **Boolean-compatible on stable endpoints**.

8.3 Structural Disjunction (Conservative Form)

```
OR_s : T5 x T5 -> T5
```

Principle

Stable TRUE dominates.

Rules

- If either operand is `S`, output is `S`
- If both operands are `Zstar`, output is `Zstar`
- Transitional dominance propagates structural uncertainty

- `Z0` remains non-collapse-valid unless paired with stable dominance

Collapse Guarantee

If `A, B ∈ {S, Zstar}`, then:

```
phi_T(OR_s(A,B)) == phi_T(A) OR phi_T(B)
```

Thus STL disjunction preserves Boolean semantics once structural stability is satisfied.

8.4 Canonical STL Axioms (Operator Discipline Framework)

The following axioms formalize STL as a conservative algebraic extension.

Axiom A1 — Determinism

For identical inputs, operators produce identical outputs:

- `NOT_s(A)` is unique for each `A ∈ T5`
- `AND_s(A,B)` and `OR_s(A,B)` are unique for each `(A,B) ∈ T5 x T5`

No randomness. No hidden state.

Axiom A2 — Closure

Operators are closed on the structural space:

- `NOT_s(A) ∈ T5`
- `AND_s(A,B) ∈ T5`
- `OR_s(A,B) ∈ T5`

No external states may emerge.

Axiom A3 — Stable Endpoint Preservation

- If `A ∈ Stable`, then `NOT_s(A) ∈ Stable`
- If `A,B ∈ Stable`, then `AND_s(A,B) ∈ Stable`
- If `A,B ∈ Stable`, then `OR_s(A,B) ∈ Stable`

Stable inputs yield stable outputs.

Axiom A4 — Non-Premature Collapse

For all `X ∈ {Z0, Eplus, Eminus}`:

```
phi_T(X) = UNDEFINED
```

Transitional or pre-structural states cannot collapse to Boolean truth.

Axiom A5 — Collapse Homomorphism on Stable Inputs

For all `A,B ∈ Stable`:

- `phi_T(NOT_s(A)) = NOT(phi_T(A))`
- `phi_T(AND_s(A,B)) = phi_T(A) AND phi_T(B)`
- `phi_T(OR_s(A,B)) = phi_T(A) OR phi_T(B)`

STL preserves Boolean algebra exactly once stability holds.

Axiom A6 — Conservative Transitional Propagation

If at least one operand is transitional and no operand is the dominating stable endpoint for that operator, the result must remain in `{Z0, Eplus, Eminus}`.

This operationalizes:

No accidental collapse.

Axiom A7 — Structural Involution of Negation

At minimum, for `A ∈ Stable`:

```
NOT_s(NOT_s(A)) = A
```

(Extended form — validated in v1.5 — holds for all `A ∈ T5`.)

Axiom A8 — Commutativity on Stable Inputs

For all `A,B ∈ Stable`:

- `AND_s(A,B) = AND_s(B,A)`
- `OR_s(A,B) = OR_s(B,A)`

Boolean parity is preserved on stable endpoints.

Axiom A9 — Associativity on Stable Inputs

For all `A,B,C ∈ Stable`:

- `AND_s(AND_s(A,B),C) = AND_s(A,AND_s(B,C))`
- `OR_s(OR_s(A,B),C) = OR_s(A,OR_s(B,C))`

Structural algebra collapses to classical Boolean algebra under stability.

Axiom A10 — Conservativity Statement

STL may introduce:

- Structural regimes
- Transitional delay
- Collapse refusal (`UNDEFINED`)

But it must **never alter classical Boolean truth once stability holds**.

---

9. Integration with SSRL (Admissibility Layer)

STL operates **after admissibility**. It does not decide whether a proposition may participate in evaluation; it governs collapse only after participation is permitted.

Layered Pipeline

For any proposition `P`:

1. `adm(P) ∈ {ALLOW, ABSTAIN}`
2. If `adm(P) = ABSTAIN`, no structural evaluation occurs.
3. If `adm(P) = ALLOW`, compute `state(P) ∈ T5`.
4. Boolean truth may be obtained only via `phi_T(state(P))`.

Layer Responsibilities

- **SSRL** governs *participation* (admissibility).
- **STL** governs *structural collapse discipline*.
- **Boolean logic** executes *terminal truth evaluation*.

This separation preserves **modular discipline**:

- No premature evaluation.
- No collapse without admissibility.
- No alteration of classical Boolean truth once stability holds.

The architecture is layered, not blended.

---

10. Deterministic Classification Template (Instantiation Framework)

To instantiate STL within any domain, a deterministic structural classifier must be defined.

Required Parameters

Each deployment must explicitly define:

- Stability window `W`
- Upper dominance threshold `tau_s`
- Lower dominance threshold `tau_l`
- Derivative threshold `eps`
- Deterministic observation vector `obs_t`

Constraints:

- `0 <= tau_l < tau_s <= 1`
- All parameters must be fixed prior to evaluation
- Parameters must be logged for replay

Structural Instantiation Requirements

Classification rules must:

- Be **finite**
- Be **deterministic**
- Be **replay-verifiable**
- Never depend on probabilistic estimation
- Never adapt thresholds dynamically
- Never use future data

Structural state assignment must be reproducible under identical inputs and parameters.

Operational Discipline

Given:

- Fixed parameter set
- Deterministic dominance metric `d_t`
- Identical observation sequence

Then:

`state_t` sequence is identical across runs.

This ensures:

- No randomness
- No training
- No model dependence
- No hidden inference

STL is executable structure, not heuristic inference.

---

11. Distinction from Fuzzy Logic

Fuzzy logic defines:

```
truth ∈ [0,1]
```

STL defines:

```
truth ∈ T5
```

Key Difference

Fuzzy logic represents **degree**.
STL represents **structural regime**.

Fuzzy logic is scalar and continuous.
STL is finite and topological.

Fuzzy logic permits partial truth.
STL permits only regime classification:

```
{Z0, Eplus, S, Eminus, Zstar}
```

There is:

- No scalar interpolation
- No probabilistic weighting
- No membership function

STL is **regime-based**, not degree-based.

It governs **collapse validity**, not truth magnitude.

---

12. Collapse Homomorphism Law

Let `op_s` be any STL structural operator corresponding to a classical Boolean operator `op_bool ∈ {NOT, AND, OR}`.

Collapse Homomorphism Principle

If all operands are stable:

```
phi_T(op_s(...)) == op_bool(phi_T(...))
```

That is, STL operators reduce exactly to Boolean operators once structural stability is satisfied.

This establishes **conservative compatibility** between STL and classical Boolean algebra.

STL may delay collapse.
STL may refuse collapse.
STL never alters classical truth once collapse is valid.

This is the compatibility theorem underlying STL's defensibility as a structural extension rather than a replacement logic.

STL therefore preserves classical logical semantics in all stable regimes while introducing a formally governed pre-collapse structural topology.

Boolean algebra remains intact; STL operates strictly as a deterministic admissibility layer preceding valid collapse.

12.1 Formal Theorem Box — Collapse Preservation & Structural Conservativity

---

Theorem 1 — Collapse Preservation

Let:

- `T5 = {Z0, Eplus, S, Eminus, Zstar}`
- `Stable = {S, Zstar}`
- `phi_T : T5 -> {TRUE, FALSE, UNDEFINED}`

For any proposition `P`:

If `state(P) ∈ Stable`, then:

```
phi_T(state(P)) == Boolean(P)
```

Therefore:

- STL does not alter classical Boolean truth.
- STL governs admissibility of collapse only.
- Classical logic is recovered exactly on stable endpoints.

Theorem 2 — Homomorphic Compatibility

Let `op_s` correspond to Boolean operator `op_bool`.

For all `A, B ∈ Stable`:

```
phi_T(op_s(A,B)) == op_bool(phi_T(A), phi_T(B))
```

Therefore:

Structural operators reduce exactly to Boolean operators under stability.

STL is a conservative structural extension of Boolean algebra.

Theorem 3 — Non-Premature Collapse Guarantee

For any `A ∈ {Z0, Eplus, Eminus}`:

```
phi_T(A) == UNDEFINED
```

Therefore:

Transitional or pre-structural states cannot produce Boolean truth.

This guarantees:

- No accidental TRUE
- No accidental FALSE
- No boundary-trigger collapse
- No threshold jitter instability

Collapse requires structural stability.

Theorem 4 — Structural Involution of Negation

For all `A ∈ T5`:

```
NOT_s(NOT_s(A)) == A
```

Additionally, for stable states:

```
phi_T(NOT_s(A)) == NOT(phi_T(A))
```

This establishes:

- Algebraic integrity of structural negation
- Compatibility with Boolean negation under collapse

Theorem 5 — De Morgan Compatibility Under Stability

For all `A, B ∈ Stable`:

```
phi_T(NOT_s(AND_s(A,B))) == NOT(phi_T(A) AND phi_T(B))
```

```
phi_T(NOT_s(OR_s(A,B))) == NOT(phi_T(A) OR phi_T(B))
```

Thus, once stability holds, STL preserves the full Boolean algebraic structure.

Together, Theorems 1–5 establish:

- Collapse preservation
- Algebraic compatibility
- Non-premature collapse discipline
- Conservative structural extension

13. What STL Is Not

STL does **not**:

- Replace Boolean logic
- Introduce probabilistic truth
- Introduce scalar or fuzzy truth
- Modify classical truth tables
- Redefine computation
- Revise physics
- Act as a predictive inference engine

STL governs **structural admissibility of collapse**.

Boolean logic remains terminal truth logic.
STL governs when terminal truth is structurally valid.

14. Architectural Positioning and System-Level Implications

Boolean logic is a **terminal-phase truth system**.

STL introduces a formally defined structural topology that precedes Boolean collapse.

Real-world systems—physical, digital, financial, and computational—evolve through **structural transition**, not instantaneous terminal truth states. Boolean evaluation remains valid and exact, but only once structural stability is satisfied.

STL therefore does not compete with Boolean logic. It operates as a **pre-Boolean structural discipline layer**, ensuring that collapse to `{TRUE, FALSE}` occurs only when structurally admissible.

In architectural terms:

- Boolean logic governs terminal truth.
- STL governs collapse validity.
- SSRL governs participation.

This layered structure preserves classical mathematics while introducing deterministic structural discipline.

15. Validation and Deployment Roadmap

STL validation proceeds through progressively stronger evidentiary layers:

Phase I — Physical Demonstrator

Deterministic structural classification of observable phase transition behavior (e.g., ice–water regime evolution), demonstrating:

- Transitional regimes produce `UNDEFINED`
- Stable endpoints collapse deterministically

Phase II — Digital Deterministic Demonstrator

Structural convergence detection using deterministic dominance metrics, validating:

- Replay-verifiable structural state evolution
- Collapse only after stability window satisfaction

Phase III — Cross-Domain Embedding

Application across structurally distinct domains, including:

- SSNT transition regimes
- SIA infinity admissibility discipline
- SSAU regime unification

Objective: demonstrate structural topology consistency independent of domain semantics.

Phase IV — Minimal Deterministic Kernel

Executable structural truth kernel with:

- O(1) per-step update cost
- O(W) memory discipline
- Full replay-verification artifacts

At this stage, STL operates as an audit-grade structural layer.

---

16. Summary Statement

Shunyaya True Logic (STL) defines a finite structural truth topology:

```
T5 = {Z0, Eplus, S, Eminus, Zstar}
```

Boolean logic remains intact.

Structural stability governs collapse validity.

Truth is not replaced.
Truth is structurally disciplined.

STL is a deterministic, conservative structural extension that:

- Preserves classical Boolean truth exactly on stable endpoints
- Prevents premature collapse in transitional regimes
- Remains algebraically compatible with Boolean operators
- Is fully replay-verifiable and domain-agnostic

It is not a new truth system.
It is a structural governance layer for truth collapse.

---

17. Verified Evidence (Replay-Verified Execution)

To ensure that STL is not merely conceptual but operationally deterministic, the STL T5 classifier has been executed and replay-verified under identical input conditions.

A deterministic input sequence was processed using fixed parameters:

- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`

Two independent executions using the same input trace produced byte-identical outputs.

Replay verification confirms:

- Structural state sequence is deterministic.
- Collapse outputs via `phi_T(state)` are deterministic.
- No randomness or hidden state influences classification.
- Identical inputs yield identical artifact hashes.

Observed replay results:

RUN1:

- `stl_trace_out.csv` =
  `a71d67ff00cbd365e1c70efe26cfc380da95d8d9513cef9e0927d135614b4736`
- `summary.txt` =
  `05a5331cf2797a1d707b7ad0c71a3f61b477245208d75012a7e6b2dce31fdc65`

RUN2:

- Hashes match exactly.

This confirms:

1. STL classification rules are fully deterministic.
2. Collapse mapping `phi_T` is stable under replay.
3. STL satisfies Shunyaya's reproducibility discipline.

Future physical demonstrators (e.g., Ice–Water protocol) will follow the same replay-verifiable artifact standard.

17.1 Verified Evidence (Controlled Ice-Like Trace)

A controlled "ice-like" dominance trace was generated deterministically and classified using the STL T5 classifier.

Parameters:

- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`

Observed structural state counts (73 rows total):

- `Z0 = 33`
- `Eplus = 9`
- `S = 11`
- `Eminus = 9`
- `Zstar = 11`

Observed collapse counts:

- `phi_T = TRUE` = 11
- `phi_T = FALSE` = 11
- `phi_T = UNDEFINED` = 51

This verifies that:

1. The classifier produces all five structural truth regimes `T5 = {Z0, Eplus, S, Eminus, Zstar}` under deterministic, controlled conditions.
2. Collapse mapping `phi_T` occurs only at stable endpoints: `phi_T(S) = TRUE` and `phi_T(Zstar) = FALSE`.
3. Transitional regimes correctly prevent premature Boolean collapse by returning `UNDEFINED`.

This controlled trace is not a claim about nature; it is a deterministic end-to-end validation of STL mechanics and collapse discipline.

17.2 Verified Evidence (Controlled Ice-Like Trace v1.1 — Nucleation Jump)

A controlled "ice-like" dominance trace (v1.1) was generated deterministically to exhibit a long pre-structural interval, a sharp "nucleation jump" into stable solid, and a sharp destabilization into stable liquid. The trace was then classified using the STL T5 classifier under fixed parameters:

- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`

Replay verification was performed by executing the same classification on the same input trace into two independent output folders. The resulting artifacts are byte-identical across runs.

Observed replay results:

RUN1:

- `stl_trace_out.csv` = `45920a2e4c3d81e59931cb2e429c17b644785741a653ff2ab9b65364cc2fe248`
- `summary.txt` = `a349a86d97d5848eddd4e0338f66ebd54e5ee88760f02640c62da96786fcaa3b`

RUN2:

- Hashes match exactly.

This confirms:

1. The end-to-end STL pipeline is deterministic under fixed inputs and parameters.
2. Structural classification into `T5` is replay-stable under repeated execution.
3. Collapse mapping `phi_T` is replay-stable and audit-grade.

This controlled trace is not a claim about nature; it is deterministic evidence that STL mechanics, stability windows, and collapse discipline execute exactly as specified.

17.3 Verified Evidence (Threshold-Edge Stress v1.2)

A threshold-edge stress trace (v1.2) was generated deterministically to test STL behavior near stability boundaries.

The trace was designed to repeatedly hover near `tau_s` and `tau_l`, intentionally crossing thresholds in single-step transitions, and then holding beyond the stability window to confirm that collapse requires stability rather than proximity.

Parameters used:

- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`

Replay verification was performed by executing the same classification on the same deterministic input trace into two independent output folders. The resulting artifacts are byte-identical across runs.

Observed replay results:

RUN1:

- `stl_trace_out.csv` = `de74326d2eb654e0008ce2f18b12448357a66085fa9c7df47f797e704bd308ac`
- `summary.txt` = `cb13074da93ee7aeee4eb8330d2732bf36bb4676316f0c93257d364cf9a41eba`

RUN2:

- Hashes match exactly.

This confirms the intended STL stability discipline:

1. Proximity to thresholds does not imply collapse validity.
2. Collapse to `TRUE` or `FALSE` occurs only when stability conditions are satisfied over window `W`.
3. Near-threshold oscillations correctly remain in transitional regimes, preventing premature Boolean collapse.
4. The full pipeline remains deterministic and audit-grade under boundary-stress conditions.

This controlled stress trace is not a claim about nature; it is deterministic evidence that STL's boundary behavior, stability requirements, and collapse discipline execute exactly as specified.

17.4 Verified Evidence (Operator Preservation v1.3)

A deterministic operator preservation test (v1.3) was executed to validate STL's collapse homomorphism law on stable endpoints.

The test enumerates all stable inputs `A,B ∈ T_stable = {S, Zstar}` and verifies executable equivalence:

- `phi_T(NOT_s(A)) == NOT(phi_T(A))`
- `phi_T(AND_s(A,B)) == phi_T(A) AND phi_T(B)`
- `phi_T(OR_s(A,B)) == phi_T(A) OR phi_T(B)`

Replay verification was performed by executing the same enumeration into two independent output folders. The resulting artifacts are byte-identical across runs.

Observed replay results:

RUN1:

- `operator_preservation_v1_3.csv` = `c767ab0c1d9d1d293158adab4bc595631c7abac679e9b2876db153912fbb2c60`
- `summary.txt` = `c240bb6647aed11535acadcc28686242fc0f1fd0156c7a970f2a49ad8bcb690a`

RUN2:

- Hashes match exactly.

This confirms:

1. STL structural operators preserve Boolean truth exactly on stable endpoints.
2. `phi_T` is a conservative collapse mapping compatible with `NOT`, `AND`, and `OR` when collapse is defined.
3. Operator preservation is executable, deterministic, and audit-grade.

17.5 Verified Evidence (Transition Propagation + Dominance Safety v1.4)

A deterministic transition propagation and dominance safety test (v1.4) was executed to validate that STL structural combinators behave conservatively in the presence of non-collapse-valid states, while still enforcing stable dominator laws where defined.

The test verifies executable properties over the full structural set `T5 = {Z0, Eplus, S, Eminus, Zstar}`:

Dominance laws:

- For all `X ∈ T5`: `OR_s(S, X) = S` and `phi_T(...) = TRUE`
- For all `X ∈ T5`: `AND_s(Zstar, X) = Zstar` and `phi_T(...) = FALSE`

No accidental collapse laws:

- For all `A,B ∈ {Z0, Eplus, Eminus}`:
  - `phi_T(AND_s(A,B)) = UNDEFINED`
  - `phi_T(OR_s(A,B)) = UNDEFINED`

Mixed stable + transitional conservative laws:

- For all `X ∈ {Z0, Eplus, Eminus}`:
  - `phi_T(AND_s(S, X)) = UNDEFINED`
  - `phi_T(OR_s(Zstar, X)) = UNDEFINED`

Replay verification was performed by executing the same enumeration into two independent output folders. The resulting artifacts are byte-identical across runs.

Observed replay results:

RUN1:

- `operator_transition_propagation_v1_4.csv` = `a43163b10aced7c6a92a86516645e150ecf8e31d4bb82dd62af41704dfeb8248`
- `summary.txt` = `ae2435e70058dde5ece7eb75fda0f94c51a15d3d52ae3dd02ec42398a1a7f985`

RUN2:

- Hashes match exactly.

This confirms:

1. STL operators prevent premature Boolean collapse when structural stability is not present.
2. Stable dominator laws are enforced deterministically and conservatively.
3. Operator behavior across stable and transitional regimes is executable, deterministic, and audit-grade.

17.6 Verified Evidence (NOT_s Involution + De Morgan v1.5)

A deterministic operator correctness test (v1.5) was executed to verify two key algebraic properties of STL:

1. NOT_s involution over the full structural truth set `T5 = {Z0, Eplus, S, Eminus, Zstar}`:

- `NOT_s(NOT_s(A)) = A` for all `A ∈ T5`

2. De Morgan compatibility on stable endpoints via the collapse mapping `phi_T`, for all `A,B ∈ {S, Zstar}`:

- `NOT(phi_T(AND_s(A,B))) == phi_T(OR_s(NOT_s(A), NOT_s(B)))`
- `NOT(phi_T(OR_s(A,B))) == phi_T(AND_s(NOT_s(A), NOT_s(B)))`

Replay verification was performed by executing the same enumeration into two independent output folders. The resulting artifacts are byte-identical across runs.

Observed replay results:

RUN1:

- `operator_demorgan_involution_v1_5.csv` = `a6fbf92ce8ec3640abecd4f3ccb66dd55e651997347bdc7e8ca3c681f9f1135e`
- `summary.txt` = `4dea7f220af6dd326cfbe96575a20038ff9e926695333035d59d95b85e98dc79`

RUN2:

- Hashes match exactly.

This confirms:

1. `NOT_s` is structurally well-formed (an involution) across all `T5`.
2. STL's conservative operators remain Boolean-compatible under collapse on stable endpoints.
3. The algebraic correctness suite is executable, deterministic, and audit-grade.

17.7 Verified Evidence (Public Dataset: CICIDS2017 DDoS Friday Row-Bin Demonstration)

STL was validated on a public intrusion-detection dataset (CICIDS2017 public distribution) using deterministic row-binning to simulate streaming operation.

Adapter definition (deterministic):

- Source file: `data\DDoS-Friday-no-metadata.parquet`
- Binning method: fixed `bin_rows = 5000`
- Dominance metric per bin: `d_t = attack_fraction_in_bin(t)`
- Output: `stl_input_t_d.csv` containing `(t, d_t, total_flows, attack_flows)`

Adapter observed totals:

- `total_rows = 221264`
- `bins = 45`
- `total_attack_flows = 128014`

Classifier execution (fixed parameters):

- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`
- `rows = 45`

Structural state results:

- `Z0 = 17`
- `Eplus = 10`
- `Eminus = 18`
- `S = 0`
- `Zstar = 0`

Collapse results:

- `TRUE = 0`
- `FALSE = 0`
- `UNDEFINED = 45`

Interpretation:

- On this streaming row-binned representation, the system exhibits persistent transitional structure (Z0/Eplus/Eminus) across all bins, and therefore STL correctly enforces `phi_T = UNDEFINED` throughout, preventing premature terminal Boolean collapse.
- This demonstrates the central STL claim: Boolean collapse is valid only after structural stability; when stability is not present, STL prevents accidental `TRUE`/`FALSE`.

Replay verification:
Two independent runs produced byte-identical artifacts.

RUN1 manifest:

- `stl_trace_out.csv` = `8c28b3fb04b33ba8f230435b9297467fdd0869d225a1a61bce632f1683d19244`
- `summary.txt` = `6b7e69a9fa3d60b0f5d973b99fdc4bb37e7c2cfca5c54c2dbaf6c318ca39d276`

RUN2 manifest matches exactly.

Dataset License and Citation:

The CICIDS2017 dataset is publicly provided by the Canadian Institute for Cybersecurity (University of New Brunswick) for research use; full licensing details, source references, and citation information are provided in Appendix F.

17.8 Verified Evidence (Public Dataset: SPX Drawdown Structural Regime Validation)

STL was validated on publicly available historical S&P 500 daily data using deterministic rolling drawdown normalization to simulate financial regime detection.

Objective

Demonstrate that Boolean collapse may be structurally premature in regime detection when stability conditions are not satisfied and that STL enforces disciplined collapse validity.

Let:

```
P = "Market is in crash regime"
```

Naive Boolean formulation:

```
Boolean(P) = TRUE if drawdown >= 0.20 else FALSE
```

STL formulation:

1. Compute normalized structural dominance:

```
d_t = min(1, drawdown_t / dd_scale)
```

where:

```
drawdown_t = (rolling_peak_252 - Close_t) / rolling_peak_252
```

- `dd_scale = 0.20`
- `rolling_peak_252` = rolling maximum over `lookback = 252` trading days

2. Classify:

```
state_t ∈ T5 = {Z0, Eplus, S, Eminus, Zstar}
```

3. Collapse only via:

```
phi_T(state_t) ∈ {TRUE, FALSE, UNDEFINED}
```

This prevents premature crash labeling during transitional drawdown expansion.

Deterministic Adapter Construction

Adapter script:

```
stl_make_d_from_spx_drawdown_v1_0.py
```

Input:

```
data\SPX_Daily.tsv
```

Output:

```
stl_input_t_d.csv
```

The adapter computes:

```
d_t ∈ [0,1]
```

Properties:

- No smoothing
- No probabilistic filtering
- No randomization
- Fully replay-verifiable transformation

Structural Parameters Used (Domain Validation Profile)

The following fixed parameters were used for this financial domain validation example:

- `lookback = 252`
- `dd_scale = 0.20`
- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`

No parameters were altered between replay runs.

These parameters apply exclusively to the SPX drawdown validation described in this section and are independent of the public conformance profile.

The authoritative public verification profile (core conformance cases) uses fixed parameters:

- `W = 20`
- `tau_s = 0.90`
- `tau_l = 0.10`
- `eps = 0.02`

Those parameters govern deterministic fingerprint generation under:

```
--profile public --cases core
```

This separation preserves a strict distinction between domain validation configurations and the deterministic public conformance profile used to generate the certified release fingerprint.

Replay-Verified Classification Runs

Classifier:

```
stl_t5_classifier_v1_0.py
```

Two independent executions were performed using identical input and parameters.

RUN1

- `stl_trace_out.csv = 10a758441e80ea1b9d561272c10d9eea51f02190c07ab848b15e71237f9f48d2`
- `summary.txt = 2c452cdcf31f0a94f38d21d426c217c588b791abf992aabac48b457f2ab8a01b`

RUN2

- Hashes match exactly.

Binary comparison confirmed:

```
FC: no differences encountered
```

`certutil -hashfile` confirms SHA-256 identity.

This establishes full replay determinism.

Structural Finding

During rising drawdown phases:

- Under a naive threshold formulation, `Boolean(P) = TRUE` if `drawdown_t >= 0.20`, producing instantaneous collapse at threshold crossing.
- STL enforces structural stability before collapse via `phi_T`.

In transitional drawdown expansion:

```
phi_T(state_t) = UNDEFINED
```

until stability is satisfied over window `W`.

This demonstrates that threshold-based Boolean formulations collapse instantaneously at boundary crossing, whereas STL requires structural stability over window `W` prior to collapse.

Cross-Domain Validation Status

STL has been deterministically executed and replay-verified across the following structurally distinct domains:

1. Phase-transition analog (ice-like structural traces)
2. Cybersecurity traffic regime detection (CICIDS2017)
3. Financial regime detection (SPX drawdown)

All runs are:

- Deterministic
- Replay-verifiable
- SHA-256 certified
- Byte-identical across executions

Structural Conclusion

The financial validation confirms:

1. STL preserves classical Boolean truth under collapse:

```
phi_T(S) = TRUE
phi_T(Zstar) = FALSE
```

2. STL enforces structural stability before collapse validity.
3. STL prevents premature regime labeling near structural thresholds.
4. STL remains fully deterministic and audit-grade.

This extends structural truth discipline into financial regime analysis without modifying classical Boolean logic.

Dataset License and Citation:

The S&P 500 daily price time series (SPX) was independently obtained from a publicly accessible financial data provider for research and educational use. Redistribution rights depend on the original provider. Full source attribution guidance and citation details are provided in Appendix F.

APPENDIX A — Collapse Homomorphism and Conservative Extension Proof

A.1 Objective

This appendix formally proves that Shunyaya True Logic (STL) is a **conservative extension** of Boolean logic.

STL introduces structural truth states prior to collapse.

STL does not modify classical Boolean truth outcomes.

A.2 Sets and Definitions

Boolean truth space:

```
B2 = {TRUE, FALSE}
```

Structural truth topology:

```
T5 = {Z0, Eplus, S, Eminus, Zstar}
```

Stable structural subset:

```
T_stable = {S, Zstar}
```

Collapse operator:

```
phi_T : T5 -> {TRUE, FALSE, UNDEFINED}
```

Defined as:

- `phi_T(S) = TRUE`
- `phi_T(Zstar) = FALSE`
- `phi_T(Z0) = UNDEFINED`
- `phi_T(Eplus) = UNDEFINED`
- `phi_T(Eminus) = UNDEFINED`

On `T_stable`, collapse is total:

```
phi_T : T_stable -> B2
```

A.3 Structural Operators

STL defines structural operators corresponding to Boolean operators:

- `NOT_s`
- `AND_s`
- `OR_s`

These operators must satisfy:

1. Determinism
2. Stable endpoint preservation
3. Collapse compatibility

A.4 Collapse Homomorphism Law

A.4.1 NOT Law

For all `A ∈ T_stable`:

```
phi_T(NOT_s(A)) = NOT(phi_T(A))
```

Proof by enumeration:

If `A = S`:

- `NOT_s(S) = Zstar`
- `phi_T(Zstar) = FALSE`
- `NOT(TRUE) = FALSE`

If `A = Zstar`:

- `NOT_s(Zstar) = S`
- `phi_T(S) = TRUE`
- `NOT(FALSE) = TRUE`

Therefore collapse compatibility holds for negation.

A.4.2 AND Law

For all `A,B ∈ T_stable`:

```
phi_T(AND_s(A,B)) = phi_T(A) AND phi_T(B)
```

Stable structural definitions:

- `AND_s(S,S) = S`
- `AND_s(S,Zstar) = Zstar`
- `AND_s(Zstar,S) = Zstar`
- `AND_s(Zstar,Zstar) = Zstar`

Boolean truth table equivalence follows directly by enumeration.

Therefore collapse compatibility holds for conjunction.

A.4.3 OR Law

For all `A,B ∈ T_stable`:

`phi_T(OR_s(A,B)) = phi_T(A) OR phi_T(B)`

Stable structural definitions:

- `OR_s(S,S) = S`
- `OR_s(S,Zstar) = S`
- `OR_s(Zstar,S) = S`
- `OR_s(Zstar,Zstar) = Zstar`

Boolean truth table equivalence follows directly.

Therefore collapse compatibility holds for disjunction.

A.5 Conservative Extension Theorem

For any Boolean operator `op_bool ∈ {NOT, AND, OR}`
and its STL structural counterpart `op_s`:

If all operands are in `T_stable`, then:

`phi_T(op_s(...)) = op_bool(phi_T(...))`

Therefore:

- STL does not alter Boolean truth values.
- STL introduces additional non-collapse states only.
- Boolean logic is exactly preserved on stable endpoints.

A.6 Interpretation

Boolean logic is terminal-phase logic.
STL introduces structural topology prior to collapse.
When stability holds, STL reduces exactly to Boolean logic.
Thus STL is a conservative structural extension of Boolean logic.

STL does not redefine truth; it governs the admissibility conditions under which truth may validly collapse.

Classical Boolean algebra remains complete and unchanged; STL operates strictly as a deterministic pre-collapse discipline layer.

APPENDIX B — Deterministic Structural Classification Template (STL Instantiation Framework)

B.1 Objective

This appendix defines how a proposition `P` is deterministically classified into the structural truth space:

```
T5 = {Z0, Eplus, S, Eminus, Zstar}
```

The goal is to ensure:

• No probabilistic interpretation
• No fuzzy scalar truth
• No learning-based inference
• Fully replay-verifiable classification
• Total deterministic rule coverage (no undefined structural states)

B.2 Structural Observation Model

At time `t`, define a deterministic observation vector:

```
obs_t = (d_t, r_t, s_t)
```

Where:

• `d_t` = dominance measure in `[0,1]`
• `r_t` = directional derivative sign
• `s_t` = stability indicator over window `W`

All quantities must be deterministically computable from current and past values only.

B.3 Required Parameters

Each domain must define:

• Stability window size `W`
• Upper dominance threshold `tau_s`
• Lower dominance threshold `tau_l`
• Derivative threshold `eps`

With constraints:

```
0 <= tau_l < tau_s <= 1
```

All parameters must be:

• Explicit
• Finite
• Fixed prior to evaluation
• Logged for replay

No adaptive thresholding is permitted.

B.4 Derived Quantities

Let:

`d_t` = dominance metric at time `t`

Examples:
• Ice-water: fraction solid
• Convergence detection: normalized residual
• Regime stability: structural occupancy ratio

Let:

```
delta_d_t = d_t - d_(t-1)
```

Define derivative sign:

If `delta_d_t > eps` → `r_t = +1`
If `delta_d_t < -eps` → `r_t = -1`
Otherwise → `r_t = 0`

Define stability deterministically:

`s_t = 1` if either `(d_t >= tau_s)` has held continuously for the previous `W` steps,
or `(d_t <= tau_l)` has held continuously for the previous `W` steps.

`s_t = 0` otherwise.

This prevents ambiguous "window drift" interpretations and ensures that structural stability is determined strictly by declared thresholds over a fixed window.

B.5 Structural Classification Rules (Total and Ordered)

Rules must be evaluated in the following strict order to guarantee determinism and totality:

Rule 1 — Stable TRUE Regime

If:

`d_t >= tau_s`
AND `s_t = 1`

Then:

`state_t = S`

Rule 2 — Stable FALSE Regime

If:

`d_t <= tau_l`
AND `s_t = 1`

Then:

`state_t = Zstar`

Rule 3 — Emerging Toward TRUE

If:

`tau_l < d_t < tau_s`
AND `r_t = +1`

Then:

```
state_t = Eplus
```

Rule 4 — Destabilizing Toward FALSE

If:

`tau_l < d_t < tau_s`
AND `r_t = -1`

Then:

```
state_t = Eminus
```

Rule 5 — Pre-Structural / Metastable

If none of the above apply:

```
state_t = Z0
```

This guarantees **total classification coverage** over all possible inputs.

This includes cases where `tau_l < d_t < tau_s` and `r_t = 0`.

B.6 Collapse Rule

Boolean projection applies only via:

```
phi_T(state_t)
```

Defined:

- `phi_T(S) = TRUE`
- `phi_T(Zstar) = FALSE`
- All others → `UNDEFINED`

No direct Boolean collapse is permitted outside `phi_T`.

B.7 Logging Schema (Replay Requirement)

Every evaluation must log:

```
t, d_t, delta_d_t, r_t, s_t, state_t, phi_T(state_t)
```

All parameters must be logged:

```
W, tau_s, tau_l, eps
```

Replay must reproduce identical structural sequences and identical artifact hashes.

B.8 Determinism Guarantee

Given:

• Fixed parameter set
• Identical observation sequence
• Deterministic arithmetic
• No future data access

Then:

The `state_t` sequence is identical across runs.

No randomness permitted.
No adaptive learning permitted.
No threshold mutation permitted.

B.9 Domain Instantiation Examples

Ice-Water Phase Change

• `d_t = fraction solid`
• `tau_s = 0.95`
• `tau_l = 0.05`
• `W = 10 seconds`
• `eps = 0.01`

Supercooling scenario:

• `d_t ~ 0`
• Structurally stable below threshold
• Classified as `Zstar` only if stability holds

• Otherwise remains transitional or `Z0`
• Collapse occurs only after stable `S`

Digital Convergence Detection

• `d_t = 1 - normalized_residual`
• `tau_s = 0.99`
• `tau_l = 0.01`

Early convergence attempt:

• `Eplus` during approach
• `S` only after stability window satisfied
• Prevents premature convergence claim

B.10 Structural Integrity Conditions

To prevent misuse:

1. Thresholds must be pre-declared.
2. Stability window must not adapt dynamically.
3. Classification must not use future data.
4. No smoothing that introduces non-determinism.
5. No hidden state outside logged parameters.

B.11 Core Structural Principle

Truth collapse requires stability.

Boolean logic assumes immediate collapse validity.

STL requires structural regime stability prior to Boolean projection.

B.12 Implementation Readiness

With this template, STL can now be:

• Applied to physical systems
• Applied to digital systems
• Embedded in deterministic kernels
• Replay-verified

• Cross-domain consistent
• Auditable at artifact level

APPENDIX C — Ice–Water Deterministic Experiment Protocol (STL Demonstrator)

C.1 Objective

Demonstrate that **Boolean collapse is structurally valid only after stability**, under a deterministic classification rule.

Proposition:

```
P = "The sample is solid."
```

We compute:

• structural state `state_t ∈ T5`
• collapse `phi_T(state_t) ∈ {TRUE, FALSE, UNDEFINED}`

We will show:

• during transition (supercooling / slush / nucleation), `phi_T = UNDEFINED`
• after stability, `phi_T` collapses cleanly to `TRUE` or `FALSE`

Important clarification:
This protocol does not revise thermodynamics.
It demonstrates structural collapse discipline under deterministic classification.

C.2 Equipment (Minimal, Safe)

• Clear cup or transparent container
• Water (distilled recommended for supercooling)
• Freezer OR ice bath
• Spoon / stirrer
• Optional: thermometer
• Timer (phone acceptable)

No hazardous materials required.

C.3 Measurement Definition (Dominance Metric)

Define:

```
d_t = f_solid(t)
```

Where:

• `d_t ∈ [0,1]`
• `d_t = 0` → fully liquid
• `d_t = 1` → fully solid

Method A — Visual Grade Scale (Controlled Rubric)

Use fixed 11-point discrete scale:

```
d_t ∈ {0.0, 0.1, 0.2, ..., 1.0}
```

You must:

• Define written rubric before experiment
• Use fixed lighting conditions
• Use same observer throughout
• Optionally record video for audit

Example rubric:

• Fully clear liquid → `0.0`
• Sparse crystals (~10%) → `0.1`
• Half slush → `0.5`
• Fully opaque solid → `1.0`

Determinism requires rubric and environment to remain fixed.

Method B — Mass Fraction (Preferred Physical Method)

If separable:

```
d_t = m_ice(t) / (m_ice(t) + m_water(t))
```

This is physically cleaner and less subjective.

C.4 Parameters (Recommended Defaults)

For baseline demonstration:

- `W = 10`
- `tau_s = 0.95`
- `tau_l = 0.05`
- `eps = 0.01`

Meaning:

- Stable TRUE requires `d_t >= 0.95` for `W` steps
- Stable FALSE requires `d_t <= 0.05` for `W` steps
- Directional transitions occur only when `delta_d_t > eps` or `delta_d_t < -eps`, as defined in Appendix B.

C.5 Sampling Schedule

Choose fixed sampling interval:

- `dt = 1s` OR `dt = 2s`

Define:

```
t = 0, 1, 2, ...
```

Log:

```
t, d_t
```

Stability window W is measured in sampling steps, not absolute time.

Optional: log `Temp(t)` (not required for STL demonstration).

C.6 Procedure Tracks

Track A — Supercooling (Strongest Demonstration)

Goal: show sudden nucleation after prolonged liquid stability.

Expected STL behavior:

- Before nucleation → `Z0` (unless `d_t <= tau_l` has satisfied stability window W, in which case Zstar)

• During rapid crystallization → `Eplus`
• After stable freeze → `S`

Collapse only after stability window satisfied.

Track B — Slush Coexistence

Goal: demonstrate prolonged transitional regime.

Expected STL behavior:

• `Eplus` or `Eminus` during drift
• `Z0` if quasi-static mixed phase
• `phi_T = UNDEFINED` until stable threshold reached

Track C — Salt Variation

Goal: demonstrate boundary dependence on structural conditions.

Two samples:

• Pure water
• Salted water

Different stability profiles confirm:

Collapse validity depends on structural regime, not single instantaneous threshold.

C.7 Input CSV Format

Header:

```
t,d
```

Example:

```
12,0.300000
```

C.8 Execution Command

```
python stl_t5_classifier_v1_0.py --in_csv your_ice_log.csv --out_dir
outputs\ICE_WATER_STL_V1\RUN1 --W 10 --tau_s 0.95 --tau_l 0.05 --eps 0.01
```

Outputs:

- `stl_trace_out.csv`
- `summary.txt`
- `MANIFEST.sha256`

Replay verification:

```
python stl_t5_classifier_v1_0.py --in_csv your_ice_log.csv --out_dir
outputs\ICE_WATER_STL_V1\RUN2 --W 10 --tau_s 0.95 --tau_l 0.05 --eps 0.01
```

Manifests must match exactly.

C.9 Acceptance Criteria

STL demonstrator is validated if:

1. Transitional interval produces
   `phi_T = UNDEFINED` for nontrivial duration
2. Stable endpoints produce
   `phi_T = TRUE` (solid stable)
   `phi_T = FALSE` (liquid stable)
3. Replay run yields identical SHA-256 hashes

This confirms deterministic structural collapse discipline.

APPENDIX D — Verified Evidence (Replay-Verified Executions)

This appendix enumerates the deterministic STL validation suite executed for this release. All executions were replayed independently and produced **byte-identical artifact bundles** under a fixed declared parameter policy `(W, tau_s, tau_l, eps)` as defined in Section 10. No parameters were altered between replay runs.

Authoritative Evidence Rule (Release Fingerprint)
For this release, evidence is bound by the **single authoritative capsule fingerprint**:

```
SHA256(REPLAY_A/MANIFEST.sha256) =
ca3c6579c0198656d6073e9dc9ce6cf6c953bf316fc17408c78c13db43e0cb44
```

This fingerprint is the SHA-256 digest of the produced `MANIFEST.sha256` for the core replay bundle and is validated by the capsule verifier.

Evidence Location (No Per-File Hash Duplication)
Per-file SHA-256 digests are **not duplicated** in this appendix to avoid drift across release rebuilds.
All artifact files, including their `MANIFEST.sha256` digests and replay summaries, are preserved under:

- `reference_outputs/` (reference evidence bundle)
- `outputs/` (local re-generation target)
- `VERIFY_STL_CAPSULE/EXPECTED_SHA256.txt` (expected manifest hash for conformance checking)

Replay equivalence is defined strictly as: `B_A = B_B`
Equality requires:

- **Byte-identical files**
- **Identical T5 classification sequences**
- **Identical operator outputs**
- **Identical collapse outputs**
- **Identical `MANIFEST.sha256` content and digest values**

No tolerance. No approximate equality. No statistical equivalence.

D.1 Digital Kernel Sanity Demonstration

The STL T5 classifier is executed on a deterministic input sequence and replay-verified.
Evidence artifacts are preserved in `reference_outputs/` and bound by the release fingerprint above.

D.2 Controlled Ice-Like Trace v1.1 (Nucleation Jump)

Deterministic dominance trace exhibiting long pre-structural phase, sharp nucleation rise, stabilization, sharp destabilization, and liquid stabilization.
Evidence artifacts are preserved in `reference_outputs/` and bound by the release fingerprint above.

D.3 Threshold-Edge Stress Test v1.2

Deterministic boundary-hovering trace verifying that proximity to thresholds does not trigger premature collapse.
Evidence artifacts are preserved in `reference_outputs/` and bound by the release fingerprint above.

D.4 Operator Preservation v1.3

Executable validation of collapse homomorphism on stable endpoints:

- `phi_T(NOT_s(A)) == NOT(phi_T(A))`
- `phi_T(AND_s(A,B)) == (phi_T(A) AND phi_T(B))`
- `phi_T(OR_s(A,B)) == (phi_T(A) OR phi_T(B))`

Evidence artifacts are preserved in `reference_outputs/` and bound by the release fingerprint above.

D.5 Transition Propagation + Dominance Safety v1.4

Executable validation that:

- Stable dominators enforce collapse
- Transitional states do not produce accidental collapse
- Mixed stable + transitional combinations remain conservative

Evidence artifacts are preserved in `reference_outputs/` and bound by the release fingerprint above.

D.6 NOT_s Involution + De Morgan v1.5

Executable validation of:

- `NOT_s(NOT_s(A)) = A` for all `A ∈ T5` (verified over the full finite domain by enumeration)
- De Morgan compatibility under collapse for stable endpoints

Evidence artifacts are preserved in `reference_outputs/` and bound by the release fingerprint above.

Appendix D Conclusion

All foundational STL tests in this appendix are:

- **Deterministic**
- **Replay-verifiable**
- **Byte-identical across independent runs**
- **Algebraically consistent**
- **Collapse-compatible with classical Boolean logic**

Certification Condition
This appendix is certified for this release if and only if:

- `VERIFY_REPLAY: PASS` and `CAPSULE_FINGERPRINT: PASS`, and
- `SHA256(REPLAY_A/MANIFEST.sha256)` matches the authoritative fingerprint listed above.

STL Deterministic Validation Matrix

ID	Test Name	Purpose	Replay Verified
D.1	Digital Kernel Sanity	Baseline T5 classification determinism	Yes (Byte-Identical Replay)
D.2	Controlled Ice-Like Trace v1.1	Structural phase transition + nucleation jump	Yes (Byte-Identical Replay)
D.3	Threshold-Edge Stress v1.2	Boundary stability enforcement (no premature collapse)	Yes (Byte-Identical Replay)
D.4	Operator Preservation v1.3	Collapse homomorphism on stable endpoints	Yes (Byte-Identical Replay)
D.5	Transition Propagation + Dominance v1.4	No accidental collapse + conservative operator safety	Yes (Byte-Identical Replay)
D.6	NOT_s Involution + De Morgan v1.5	Algebraic correctness of NOT_s and De Morgan compatibility	Yes (Byte-Identical Replay)

Validation Properties Confirmed

The above test suite confirms that STL:

1. Is fully deterministic and replay-verifiable.
2. Preserves classical Boolean truth under collapse `phi_T`.
3. Enforces structural stability before collapse validity.
4. Prevents premature truth assignment near thresholds.
5. Propagates transitional regimes conservatively.
6. Satisfies algebraic integrity (involution + De Morgan) on stable endpoints.

Appendix D Certification Statement

All tests in Appendix D were executed under fixed parameters and replayed independently, producing byte-identical artifacts across runs.

Replay equivalence condition satisfied:

```
B_A = B_B
```

The core structural, algebraic, and collapse-preservation properties of STL have therefore been deterministically validated under executable replay:

• Structurally sound
• Collapse-compatible
• Algebraically consistent
• Deterministic by construction

APPENDIX E — Structural Admissibility Analysis (Maturity & Deployment Discipline Layer)

E.1 Purpose

This appendix elevates STL from “working and replay-verified” to a maturity-level structural framework by documenting:

1. Parameter sensitivity envelope (bounded variation discipline)
2. Quantified Boolean-vs-STL collapse lag (Structural Admissibility Delay)
3. Failure modes and misuse boundaries
4. Scalability characteristics

This appendix does not modify STL.
It formalizes the structural conditions under which STL remains disciplined, interpretable, and reproducible.

E.2 Core Definitions

E.2.1 Structural Truth Space

T5 = {Z0, Eplus, S, Eminus, Zstar}

E.2.2 Collapse Mapping

phi_T : T5 -> {TRUE, FALSE, UNDEFINED}

• phi_T(S) = TRUE
• phi_T(Zstar) = FALSE
• phi_T(Z0) = UNDEFINED
• phi_T(Eplus) = UNDEFINED
• phi_T(Eminus) = UNDEFINED

Principle:
Boolean truth is valid only after structural stability.

E.2.3 Stability Window

Collapse is permitted only if dominance remains stable over W consecutive steps.

Stable TRUE condition:
d_t >= tau_s for W steps

Stable FALSE condition:
d_t <= tau_l for W steps

E.2.4 Dominance Derivative Threshold

delta_d_t = d_t - d_(t-1)

A step is materially transitional if:

abs(delta_d_t) > eps

This governs transitional regimes (Eplus, Eminus).

E.3 Parameter Sensitivity Envelope

E.3.1 Baseline Parameters Used in Verified Runs

• W = 10
• tau_s = 0.95
• tau_l = 0.05
• eps = 0.01

E.3.2 Bounded Variation Discipline

STL remains conservative provided:

• tau_s remains near 1.0
• tau_l remains near 0.0
• W is not trivially small
• eps does not suppress real transitions

E.3.3 Recommended Public Validation Envelope

For external evaluators:

• W ∈ {5, 10, 20}
• tau_s ∈ {0.90, 0.95, 0.99}
• tau_l ∈ {0.10, 0.05, 0.01}
• eps ∈ {0.005, 0.01, 0.02}

Deterministic effects:

• Smaller W → faster collapse (closer to threshold Boolean behavior)
• Larger W → slower collapse (more conservative gating)
• Relaxed thresholds → more frequent collapse
• Tightened thresholds → stricter admissibility

Governance rule:
Every deployment must publish:

• parameter set
• dominance metric definition d_t
• replay-verified hashes

Legitimacy requires transparency.

E.4 Structural Admissibility Delay (SAD)

E.4.1 Formal Definitions

To eliminate ambiguity, STL defines two complementary admissibility metrics.

1. Event-Based Structural Admissibility Rate

Let:

• `E_bool` = set of all naive Boolean threshold events
• `E_prem` = subset of `E_bool` where Boolean collapses at `t_bool` but STL does **not** collapse to the new Boolean truth at `t_bool`

Define:

```
SAD_rate(P) = |E_prem| / |E_bool|
```

Properties:

• `0 <= SAD_rate(P) <= 1`
• `SAD_rate(P) = 0` → no premature Boolean collapses
• `SAD_rate(P) = 1` → all Boolean collapses were structurally premature

Interpretation:

`SAD_rate(P)` measures the **proportion of Boolean collapses that STL rejects at the threshold instant**.

This is the primary cross-domain comparability metric.

---

2. Timing-Based Structural Admissibility Delay

For each Boolean threshold event `i`:
Let:
• `t_bool(i)` = time of the `i`-th Boolean threshold event
• `Bool_i` = Boolean truth value produced by the naive Boolean trigger at `t_bool(i)`
• `t_stl(i)` = first time `t >= t_bool(i)` where `phi_T(state_t) = Bool_i`

Define:
If `t_stl(i)` exists:
`SAD_delay(i) = t_stl(i) - t_bool(i)`

If `t_stl(i)` does not exist within observed horizon:
`SAD_delay(i)` is **undefined** (structural refusal).

Summary reporting may include:
• Count of `SAD_delay(i) > 0`
• Mean / median of positive delays
• Count of never-collapse events

Interpretation:
• `SAD_delay(i) > 0` → STL enforced additional stability
• `SAD_delay(i) = 0` → simultaneous collapse
• Undefined → STL refused collapse entirely

`SAD_delay` quantifies temporal enforcement strength, while `SAD_rate` quantifies frequency of premature Boolean collapse.

E.4.2 Reporting Template

For each dataset, publish:

• Proposition `P`
• Naive Boolean trigger definition
• `E_bool`
• `E_prem`
• `SAD_rate(P)`
• (Optional but recommended) timing table for `SAD_delay(i)`
• Parameter set: `W`, `tau_s`, `tau_l`, `eps`
• Replay artifact hashes

This converts structural discipline into measurable, audit-grade comparison.

E.5 Failure Modes and Misuse Boundaries

E.5.1 Never-Collapse

If thresholds are too strict or `W` too large:

• `phi_T(state_t)` may remain `UNDEFINED`

This is structural refusal, not malfunction.

STL intentionally prioritizes admissibility over forced decision.

E.5.2 Boolean Degeneration

If thresholds are too relaxed or `W` too small:

• STL degenerates toward threshold Boolean logic
• Structural gating effect disappears
• `SAD_rate(P)` approaches 0

Therefore parameter publication is mandatory.

E.5.3 Adapter Misdefinition

STL correctness depends on a meaningful deterministic dominance metric:

```
d_t ∈ [0,1]
```

If `d_t` is poorly defined, STL reflects adapter noise rather than structural regime.

Adapters must be:

• deterministic
• bounded
• documented
• replay-verifiable

Artificial oscillation engineered to inflate `SAD_rate(P)` invalidates structural claims.

E.5.4 Automated Control Boundary

STL is a structural admissibility layer, not an autonomous actuator.

It is suitable for:

• observability
• regime gating
• audit-grade structural discipline

It is not a replacement for domain-specific safety systems.

E.6 Scalability Characteristics

E.6.1 Per-Step Computational Cost

For streaming `d_t`:

• Update cost per step: `O(1)`
• Memory cost: `O(W)`

No training.
No probabilistic inference.
No optimization loops.

E.6.2 Replay Discipline

Each run must produce:

• trace CSV
• summary TXT
• `MANIFEST.sha256`

Maturity-level credibility requires deterministic reproducibility.

E.7 Maturity Criteria

STL reaches maturity when:

1. Algebraic integrity suite passes
2. Replay verification passes (byte-identical artifacts)
3. At least one public dataset demonstrates `SAD_rate(P) > 0`
4. `SAD_rate(P)` is computed and reported
5. Failure modes are documented
6. Parameters and adapters are published

Advanced maturity includes structured publication of `SAD_delay` tables.

All criteria except formal public SAD tables are already satisfied.

The final maturity artifact is structured, replay-certified SAD reporting for the public datasets.

Appendix E Conclusion

STL is:

• Deterministic
• Replay-verifiable
• Algebraically conservative
• Cross-domain validated
• Boolean-compatible under collapse

With dual SAD metrics:

• `SAD_rate(P)` — proportional premature collapse mitigation
• `SAD_delay(i)` — quantified stability enforcement lag

Structural admissibility becomes:

• measurable
• auditable
• cross-domain comparable
• replay-certified

This elevates STL from conceptual conservativity to quantified structural discipline.

APPENDIX F — Public Dataset Licenses and Citations

This appendix enumerates all external public datasets used for STL validation.
These datasets are not owned by the Authors of STL and are referenced strictly for research, reproducibility, and educational illustration.

STL validation does not modify, redistribute, or alter the original datasets.
Users are responsible for verifying current licensing terms with the original data providers.

No dataset files are distributed within this repository.

F.1 CICIDS2017 — Intrusion Detection Dataset

Dataset Description

CICIDS2017 — Canadian Institute for Cybersecurity Intrusion Detection Dataset (2017)

The dataset contains labeled network flow records representing benign and attack traffic collected across multiple days.

Used in STL validation to demonstrate structural collapse discipline under deterministic row-binning.

Source

Provided by:
Canadian Institute for Cybersecurity (CIC)
University of New Brunswick

Kaggle mirror (curated Parquet distribution):
https://www.kaggle.com/datasets/dhoogla/cicids2017?resource=download

---

Original Publication

Imran Sharafaldin, Arash Habibi Lashkari, and Ali A. Ghorbani
“Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization”
Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP), 2018.

Recommended Citation (BibTeX)

```
@inproceedings{sharafaldin2018toward,
title={Toward generating a new intrusion detection dataset and intrusion traffic
characterization},
author={Sharafaldin, Imran and Lashkari, Arash Habibi and Ghorbani, Ali A},
booktitle={2018 4th International Conference on Information Systems Security and Privacy
(ICISSP)},
pages={108--116},
year={2018},
organization={IEEE}
}
```

License Information

CICIDS2017 is released for research and educational use by the Canadian Institute for Cybersecurity.

Users must:

- Acknowledge the dataset source in derivative works.
- Review the official CIC website for updated licensing terms.
- Ensure compliance with institutional and commercial usage restrictions.

Usage Notes for STL Validation

- The Kaggle mirror provides curated Parquet files retaining flow features and labels.
- Full PCAP metadata is not included.
- STL evaluation used deterministic row-binning to construct structural dominance metrics.
- No probabilistic sampling, smoothing, or model training was applied.

F.2 S&P 500 Index — Daily Close Dataset

Dataset Description

S&P 500 Index — Historical Daily Price Time Series

Contains daily Open, High, Low, Close, and Volume values across historical periods.

Used in STL validation to demonstrate structural collapse discipline in financial regime detection using deterministic rolling drawdown normalization.

Source

Data retrieved from a publicly accessible market data provider (e.g., Stooq.com or equivalent public source).

The dataset was independently downloaded by the user.

Example public provider:

https://stooq.com/

Neutral Citation Template
S&P 500 Index — Daily Close Time Series.
Historical daily price data independently obtained from a public market data provider (e.g., Stooq or equivalent). Accessed for research and educational purposes.

License Information

The S&P 500 historical price data referenced during STL validation was independently retrieved from a publicly accessible provider.

However:

- Redistribution terms depend on the original provider.
- Commercial reuse may require explicit permission.
- Users must consult the official provider website for current licensing conditions.

STL validation uses the dataset solely for deterministic structural evaluation and research illustration.

Usage Notes for STL Validation

- STL adapter computed rolling drawdown:

  ```
  drawdown_t = (rolling_peak_252 - Close_t) / rolling_peak_252
  ```

- Dominance normalization:

  ```
  d_t = min(1, drawdown_t / dd_scale)
  ```

- Parameters:

  ```
  lookback = 252
  dd_scale = 0.20
  ```

- No predictive modeling was performed.
- No financial advice is implied.
- The dataset is used strictly to demonstrate structural collapse discipline.

Appendix F Conclusion

All public datasets referenced in STL validation:

- Are externally sourced.
- Remain property of their original providers.
- Were used strictly for deterministic structural evaluation.
- Were not altered beyond documented deterministic transformations.
- Require users to independently verify current licensing terms.

This maintains:

- Legal clarity
- Citation integrity
- Structural reproducibility
- Professional publication standards

APPENDIX G — Structural Admissibility Quantification (SAD)

G.1 Objective

Replay verification establishes:

• Determinism
• Algebraic correctness
• Collapse compatibility

However, civilization-grade maturity requires an additional layer:

Quantified structural admissibility reporting.

STL defines Structural Admissibility as a measurable indicator of how often — and how long — STL prevents premature Boolean collapse at threshold events.

Two complementary metrics are defined:

• `SAD_rate(P)` — proportional premature-collapse mitigation
• `SAD_delay(i)` — timing-based stability enforcement

G.2 Formal Definitions

Let:

• `P` = proposition under evaluation

• `E_bool` = set of indexed naive Boolean threshold events
(i.e., all times `t_bool(i)` at which the naive Boolean rule changes truth value)

• `Bool_i` = Boolean truth value produced by the naive rule at `t_bool(i)`

• `E_prem` ⊆ `E_bool` = subset of events `i` such that
`phi_T(state_{t_bool(i)}) != Bool_i`
(i.e., STL does not collapse to the new Boolean truth at the threshold instant)

G.2.1 Event-Based Structural Admissibility Rate

Define:

```
SAD_rate(P) = |E_prem| / |E_bool|
```

Where:

• `|E_bool|` = total number of naive Boolean threshold events
• `|E_prem|` = number of those events where STL does not collapse to the new Boolean truth at `t_bool` (i.e., STL is `UNDEFINED` or opposite at the threshold instant)

Properties:

• `0 <= SAD_rate(P) <= 1`
• `SAD_rate(P) = 0` → No premature Boolean collapse
• `SAD_rate(P) = 1` → All Boolean collapses were structurally premature

Interpretation:

`SAD_rate(P)` measures the **proportion of Boolean collapses that were structurally premature**.

This is the primary cross-domain comparability metric.

G.2.2 Timing-Based Structural Admissibility Delay

For each Boolean threshold event `i`:

Let:

- `t_bool(i)` = time of i-th Boolean threshold event
- `t_stl(i)` = first time at or after `t_bool(i)` when `phi_T(state_t)` collapses to the same Boolean truth

Define:

If `t_stl(i)` exists:

```
SAD_delay(i) = t_stl(i) - t_bool(i)
```

If `t_stl(i)` does not exist:

Structural refusal is recorded (no collapse).

Interpretation:

- `SAD_delay(i) > 0` → STL enforced additional stability
- `SAD_delay(i) = 0` → simultaneous collapse
- Undefined → STL refused collapse

`SAD_delay(i)` measures **temporal enforcement strength**, while `SAD_rate(P)` measures **frequency of premature Boolean collapse**.

G.3 Interpretation Bands (Event-Based)

The following bands apply to `SAD_rate(P)`:

SAD_rate(P)	Interpretation
0.00	No structural safety advantage
0.01–0.25	Minor premature collapse mitigation
0.26–0.50	Moderate structural protection
0.51–0.75	Strong collapse discipline
0.76–1.00	High structural admissibility enforcement

Civilization-grade systems in threshold-sensitive domains should typically demonstrate:

```
SAD_rate(P) > 0.50
```

G.4 SAD Measurement Protocol

For any dataset:

1. Define proposition `P`
2. Define naive Boolean threshold condition
3. Identify all Boolean threshold events `E_bool`
4. For each event at time `t_bool`, check whether STL collapses to the new Boolean truth at `t_bool`
5. Count premature events `E_prem`
6. Compute: `SAD_rate(P) = |E_prem| / |E_bool|`

Optional (recommended when time index is meaningful):

7. Compute `SAD_delay(i)` for each event

All steps must be:

• Deterministic
• Replay-verifiable
• Hash-certified

No probabilistic sampling is permitted.

G.5 Example – Financial Domain (SPX)

Proposition:

```
P = "Market in crash regime"
```

Naive Boolean:

`Boolean(P) = TRUE` iff `drawdown >= 0.20`

STL:

• Collapse only after stability window `W` satisfied
• Transitional states return `UNDEFINED`

Observation:

• Boolean collapses immediately at threshold crossing
• STL delays collapse until structural stability

Result:

```
SAD_rate(P) > 0
```

(Exact quantified value to be reported via structured replay-certified release.)

G.6 Example – Cybersecurity Domain (CICIDS)

Proposition:

```
P = "Traffic bin is malicious"
```

Naive Boolean:

`Boolean(P) = TRUE` iff `attack_fraction_in_bin >= threshold`

Observed:

• Boolean threshold events: `|E_bool| = 45`
• Premature events: `|E_prem| = 45`

Therefore:

```
SAD_rate(P) = 45 / 45 = 1.00
```

Interpretation:

All Boolean collapses were structurally premature under the evaluated binning configuration.

This indicates full structural admissibility enforcement.

G.7 Example – Negative Control (NEGCTL Debounced Trace)

Purpose:

A deterministic negative control ensures SAD reporting is not an artifact of noisy real-world data.

Dataset:

NEGCTL Debounced Trace (Corner Sweep)

Naive Boolean:

`Boolean(P) = TRUE` iff `d_t >= 0.50`

Observed (example case):

```
E_bool = 1
E_prem = 1
SAD_rate(P) = 1.000000
```

Interpretation:

• The naive Boolean crosses the threshold instantly.
• STL enforces stability gating and does not collapse at `t_bool`.
• The Boolean collapse is counted as structurally premature.

This confirms:

• SAD computation matches the formal definition.
• The reporting layer is deterministic and definition-faithful.

G.8 Cross-Domain Structural Maturity Table

Domain	Boolean Thresholding	STL Collapse Discipline	Replay Verified	SAD Status
Phase Transition (Controlled)	Instantaneous	Stability-gated	Yes	High
Cybersecurity (CICIDS)	Immediate per bin	Fully stability-gated	Yes	Very High
Financial (SPX)	Threshold-instant	Stability-disciplined	Yes	Moderate–High
Negative Control (NEGCTL)	Threshold-instant	Stability-gated	Yes	Definition-verified

---

G.9 Structural Conclusion

With quantified SAD metrics:

1. Structural admissibility becomes measurable.
2. Collapse discipline becomes auditable.
3. Cross-domain safety margins become comparable.
4. Boolean logic remains intact — but structurally governed.

This completes the maturity ladder:

Concept → Determinism → Algebra → Replay Verification → Quantified Structural Discipline

At this stage:

STL is not merely theoretical.
It is structurally measurable.

APPENDIX H — Structural Adapter Proof Obligations (Cross-Domain Instantiation Contract)

H.1 Purpose

This appendix defines the minimum proof obligations any domain adapter must satisfy for an STL instantiation to be:

- Deterministic
- Replay-verifiable
- Conservative under collapse
- Resistant to metric manipulation or parameter tuning abuse

STL guarantees algebraic conservativity.
This appendix governs semantic admissibility.

H.2 Layer Separation Principle

Every STL deployment has two layers:

1. **Adapter Layer**
   Transforms domain observations into `d_t ∈ [0,1]`
2. **Classifier Layer**
   Maps `d_t` into `state_t ∈ T5` using fixed parameters

The adapter defines meaning.
The classifier enforces structural discipline.

All proof obligations apply to the Adapter Layer.

H.3 Adapter Contract (Mandatory Disclosure)

For proposition `P`, publish:

1. Dominance metric definition (`d_t ∈ [0,1]`)

2. Sampling policy (fixed `dt`)
3. Parameters: `W`, `tau_s`, `tau_l`, `eps`
4. Naive Boolean trigger definition
5. Logging schema:
   `t, d_t, delta_d_t, r_t, s_t, state_t, phi_T(state_t)`
6. Replay evidence (MANIFEST hashes from ≥2 independent runs)

Without these disclosures, SAD reporting is invalid.

H.4 Proof Obligations

PO1 — Boundedness
`0 ≤ d_t ≤ 1` for all t.

PO2 — Deterministic Computability
Identical raw input → identical outputs.

No randomness.
No hidden state.
No probabilistic filtering.
No training.

PO3 — No Future Leakage
`state_t` depends only on observations up to time `t`.

PO4 — Parameter Freeze
`W`, `tau_s`, `tau_l`, `eps` declared prior to run and remain constant.

PO5 — Collapse Conservativity
Stable endpoints align with classical truth:

```
phi_T(S) = TRUE
phi_T(Zstar) = FALSE
```

PO6 — Metric Interpretability Obligation

The dominance metric `d_t` must:

1. Be physically or semantically meaningful
2. Be monotonic relative to proposition `P`
3. Not be engineered to inflate SAD

Artificial transitional oscillations invalidate structural claims.

H.5 External Review Checklist

Publishable only if:

- PO1–PO6 documented
- Two replays match byte-for-byte
- Boolean rule explicitly stated
- Parameters frozen and logged
- Dominance metric justified

H.6 Structural Integrity Statement

STL ensures algebraic conservativity.
The Adapter Contract ensures semantic discipline.

Together they guarantee:

- Structural collapse is meaningful
- SAD reporting is legitimate
- Cross-domain comparison is defensible

APPENDIX I — SAD(P) Reporting Format (Audit-Grade Publication Layer)

I.1 Purpose

This appendix defines the exact reporting tables required for civilization-grade publication of `SAD(P)`, ensuring cross-domain comparability, auditability, and replay-verifiable transparency.

All reported values must be derived deterministically from replay-certified artifacts.

I.2 Required Table 1 — Dataset / Run Declaration

For each evaluated run, publish:

- Dataset name
- Dataset source
- Adapter name
- Proposition `P`

- Naive Boolean rule for `P`
- Parameters: `W, tau_s, tau_l, eps`
- Output artifact hashes (CSV, summary, MANIFEST)

This establishes:

- Reproducibility
- Parameter freeze compliance

I.3 Required Table 2 — Event-Level Structural Admissibility Accounting

Define:

• `E_bool` = total number of naive Boolean threshold events

• `E_prem` = number of events where Boolean collapses at `t_bool` but STL does **not** collapse to the new Boolean truth at `t_bool`
• `E_aligned` = number of events where STL collapse aligns with Boolean at `t_bool`

Publish:

• `SAD_rate(P) = E_prem / E_bool`
• `E_bool`
• `E_prem`
• `E_aligned`

Required consistency condition:

`E_bool = E_prem + E_aligned`

Properties:

• `0 <= SAD_rate(P) <= 1`
• `SAD_rate(P) = 0` → No premature Boolean collapse
• `SAD_rate(P) = 1` → All Boolean collapses were structurally premature

Interpretation:

`SAD_rate(P)` measures the **proportion of Boolean collapses that were structurally premature**.

All counts must be computed deterministically from the same replay-certified trace used for artifact hashing.

This table is mandatory for civilization-grade reporting and must match the formal definition in Appendix G.

I.4 Required Table 3 — Timing-Based Structural Delay (Optional but Recommended)

If the domain supports a meaningful ordered index (time, step, bin number), also publish per-event timing evidence.

For each Boolean threshold event `i`, publish:

• `t_bool(i)` = time (or index) of i-th Boolean threshold event
• `t_stl(i)` = first time at or after `t_bool(i)` when STL collapses to the new Boolean truth (if it ever does)
• `SAD_delay(i) = t_stl(i) - t_bool(i)` when `t_stl(i)` exists

If STL never collapses for that event:

• Mark as `REFUSAL` (no collapse)

Publish summary statistics:

• Count of events where `SAD_delay(i) > 0`
• Mean of positive `SAD_delay(i)`
• Median of positive `SAD_delay(i)`
• Count of `REFUSAL` events

This table provides **temporal enforcement evidence** in addition to proportional `SAD_rate(P)`.

Together:

• Table I.3 quantifies frequency of premature Boolean collapse
• Table I.4 quantifies duration of enforced structural stability

Both remain fully deterministic and replay-verifiable.

I.5 Publication Rule (Non-Negotiable for Civilization-Grade Verification)

A `SAD(P)` claim is publishable only if:

1. Tables **I.2** and **I.3** (and **I.4** when timing is meaningful) are provided.
2. Output artifacts are replay-verified.
3. SHA-256 hashes are included for independent verification.
4. Boolean trigger definition is explicitly stated.
5. Parameter set is declared and frozen.

Without these disclosures, `SAD(P)` claims are not considered civilization-grade publishable.

I.6 Negative Control Requirement (Civilization-Grade Strengthening)

For civilization-grade maturity, publish at least one deterministic negative control run, with:

- Two independent replays (`RUN1`, `RUN2`)
- Byte-identical SAD outputs (or identical `MANIFEST.sha256` hashes)
- A clear proposition and naive Boolean rule

This demonstrates:

- The reporting layer is deterministic,
- The SAD computation is definition-faithful, and
- Claims are resistant to accidental tool drift.

Appendix J — SAD Quantification Audit Seal

J.1 Purpose

This appendix documents the audit sealing of Structural Admissibility Quantification (SAD(P)) results across validated STL domains.

The purpose of this audit seal is to confirm that:

1. SAD(P) tables are deterministically generated.
2. RUN1 and RUN2 outputs are byte-identical.
3. Quantification preserves conservative collapse semantics.
4. No modification to STL core logic was required.

The audit seal functions as an integrity layer.
It does not introduce new logic.
It confirms reproducibility of existing results.

J.2 Scope of Sealed Artifacts

The sealed audit capsule contains the following artifacts:

- CICIDS SAD(P) tables (RUN1 and RUN2)
- NEGCTL SAD(P) tables (RUN1 and RUN2)
- SPX SAD(P) tables (RUN1 and RUN2)
- Per-run `MANIFEST.sha256` files
- A master capsule manifest enumerating all sealed files

Total sealed files: 30
(3 domains × 2 runs × 5 files per run)

All files were:

- Enumerated deterministically
- Hashed under SHA256
- Verified for cross-run parity
- Confirmed byte-identical across replay

The capsule exists as an internal audit artifact.

J.3 Conservative Preservation

SAD quantification does not alter Boolean logic.

All evaluation remains conservative under:

```
phi((m,a,s)) = m
```

Where:

- `m` = classical Boolean result
- `a` = admissibility posture
- `s` = structural state

SAD(P) measures structural posture only.
It does not modify truth values.
It does not alter collapse semantics.
It does not redefine logical operators.

The audit seal confirms preservation of classical behavior.

J.4 Replay Identity Confirmation

For each validated domain:

- RUN1 == RUN2 (byte-identical)
- Per-run manifests verified
- Cross-run parity confirmed
- Deterministic parameters fixed

No probabilistic methods were used.
No adaptive thresholds were used.
No model inference was used.

Replay verification confirms deterministic reproducibility within the defined STL scope.

J.5 Master Capsule Hash

The internal SAD audit capsule is sealed by the following SHA256 digest:

`d36de22b5882b6d1a55583cb84868de5ade711a19048ebc3c5b3c66795fdd241`

This digest uniquely represents:

• The full set of sealed SAD artifacts
• Their deterministic enumeration
• Their byte-level integrity

Any modification to any sealed file would produce a different digest.

The digest serves as a cryptographic integrity anchor for the quantified results.

J.6 Public Audit Capsule

The public `STL_SAD_CAPSULE` folder contains:

• `CAPSULE_MANIFEST.sha256`
• `README_CAPSULE.txt`

The capsule does not duplicate SAD tables.

It provides a master SHA256 digest that binds the complete set of SAD(P) artifacts generated during deterministic STL execution.

The SAD tables themselves are available within the STL outputs directory.

Independent verification may be performed by:

1. Recomputing SHA256 hashes for the publicly released SAD files.
2. Confirming equality with the values recorded in the capsule manifest.
3. Confirming cross-run replay identity (RUN1 == RUN2).

If hashes match, quantification integrity is confirmed.
The capsule functions as an integrity anchor, not a data mirror.

J.7 Independent Verification Principle

To independently validate SAD integrity:

1. Recompute SHA256 for each published SAD file.
2. Confirm equality with the corresponding per-run `MANIFEST.sha256`.
3. Confirm RUN1 and RUN2 equality.
4. Confirm consistency with the master capsule digest above.

If hashes match, deterministic quantification integrity is confirmed.

J.8 Civilization-Grade Interpretation (Within STL Scope)

Within STL context, civilization-grade refers to:

- Deterministic execution
- Conservative algebraic extension
- Replay-verifiable artifacts
- Explicit parameter disclosure
- Hash-sealed quantification
- Non-interference with classical logic

The audit seal establishes the structural maturity chain within the current STL scope:

Concept

→ Determinism

→ Algebraic Proof

→ Replay Identity

→ Cross-Domain Validation

→ SAD Quantification

→ Hash-Sealed Audit Confirmation

This confirms deterministic reproducibility and quantification integrity within the defined STL boundaries.

Final Notes

This appendix:

- Does not introduce new logic
- Does not redefine Boolean semantics
- Does not alter STL operators
- Does not expand scope
- Does not imply regulatory certification

It documents integrity discipline.

OMP