

RESEARCH LIBRARY

Operator-Symbolic Language

Proof-carrying interfaces

Traditional paper

Komorebi P.

Research edition 1.0 / OSL-Core 0.5 / Standardized 8 October 2026

Reading scope

Finalized object. A consolidated language manuscript, a frozen OSL-Core 0.5 reference implementation, the finite Discovery Bridge, and a checked finite-state interface. The edition number is not a new runtime version. Neural proposals, formal statements, operational permissions, and historical evidence retain different types and different assurance levels.

Abstract

A machine-oriented language should specify what an expression does, what information it depends on, and under which conditions another system may use it. Operator-Symbolic Language (OSL) develops this principle as an explicit intermediate representation for heterogeneous computational agents. Its geometric layer uses typed state spaces, operators, and structure-preserving transport. Its operational layer distinguishes access, provenance, endorsement, proof resources, memory, planning, bounded capabilities, transactions, and finite communication protocols. Its evidence layer separates receipt integrity, source-bound semantic checking, interpreter replay, and complete finite enumeration.

This edition consolidates the earlier OSL v0.5 monograph and its OSL-Core 0.3 implementation with Core 0.4 receipt hardening, Core 0.5 semantic independence, and the finite Discovery Bridge. Fresh execution reproduces the Core 0.5 corpus exactly: 66 programs, 1,825 receipts, and all recorded acceptance or rejection outcomes. Its 168-test suite passes. The Discovery Bridge's 20 tests and three generated programs also pass against the current core. A new, deliberately finite interface constructs and checks continuation-preserving quotients, serializes semantic checkpoints, and routes accepted or unverified candidates through the actual OSL runtime.

The result is an executable research language, not a general-purpose proof assistant, a production security boundary, or a demonstration that a neural model has spontaneously learned OSL. The paper gives the mathematical contracts needed for the language, identifies the portions implemented and checked, and preserves the failed or absent learned-language evidence. The stable interface is explicit semantics; latent computation remains a separately tested producer of candidate representations.

How to read the evidence. "Proved" means a mathematical derivation under stated assumptions. "Replayed" means a computation completed in this edition. "Retained" means a historical result was recovered and its source identified. Neither unit-test counts nor archive hashes turn a retained experiment into an independent scientific replication. The original 88-page monograph and all recovered OSL experimental lineages remain available in the accompanying source archives.

Contents

1 The language problem 3
2 Objects, meanings, and realizations 3
3 Operator transport and compositional semantics 4
4 The operational state machine 6
5 Receipt integrity and independent checking 7
6 Bounded discovery and semantic admission 8
7 Worked OSL programs 10
8 A checked semantic-checkpoint interface 11
9 What the retained neural-language experiments establish 12
10 Fresh verification and reproducibility 12
11 Compatibility, trust, and publication status 14
A Claim register 15
B Source hierarchy and reading map 15
C A compact operational checklist 16

1 The language problem

The phrase *machine-native language* can denote three different objects: a learned latent representation, a notation optimized for machines, or an intermediate representation with an independently specified semantics. OSL addresses the third. It is intended to make heterogeneous internal computations communicate through explicit operations, claims, and obligations. It does not require every producer to use the same coordinates, nor does it require a producer's internal state to resemble a string in the external language.

Consider an agent that receives a vector from another agent. Knowing the vector's dimensions is not enough. The recipient may need to know which operation produced it, which coordinate system it inhabits, which observations supported it, whether a transformation preserved its meaning, and whether it may be used for a particular action. A fluent explanation is not a substitute for those relations. OSL makes them separate parts of the program state.

The proposal has established neighbors. Abstract interpretation formalizes sound abstraction of computation; proof-carrying code separates untrusted production from policy-relative checking; session types make communication protocols part of a language's discipline. OSL draws on these ideas rather than claiming to replace them. Its specific research object is their integration with operator transport, provenance-sensitive memory, and explicit neural admission boundaries. [CC77, N97, HYC08]

The present release should also be distinguished from an argument about conscious machines. No linguistic syntax, proof receipt, or recurrent state establishes experience. The operational question is narrower: can one system offer a reusable transformation or claim that another system can interpret and check without trusting the producer's narrative?

1.1 What has been finalized

The manuscript fixes terminology and scope, identifies one current executable core, and binds its claims to reproducible artifacts. OSL-Core remains version 0.5 with receipt schema `osl.core.receipt.v2`. The language's broader mathematical specification remains available in the preserved v0.5 monograph. The Discovery Bridge remains a separately versioned bounded synthesizer. Experimental packages numbered v0.6 or v0.3.1 are not later editions of the language merely because one number is larger. [O03, O05, ODB]

This is consequently a *research-edition freeze*. Future runtime changes should receive their own version, source manifest, compatibility statement, and new evidence. An unfinished historical Core 0.6 development mentioned in prior working records is not promoted into a released implementation here: no complete authoritative successor was recovered.

2 Objects, meanings, and realizations

2.1 Three distinct interfaces

We use the following separation throughout:

$$\text{latent producer} \longrightarrow \text{typed proposal} \longrightarrow \text{checked operational consumer.}$$

The latent producer can be a neural network, a search procedure, a human, or a fixed fixture. The typed proposal names a carrier, operation, scope, and evidence. The consumer decides what the proposal

permits within an explicit language semantics. Success at one arrow does not establish success at the other.

Definition 2.1 (Realization)**.** A realization of an operation vocabulary $\mathcal{G}$ consists of a state space X, maps $\rho_X(g) : X \to X$ for $g \in \mathcal{G}$, and a declared observation map $O_X : X \to Y$. For typed operations, each generator instead has an explicit source and target carrier, and only type-compatible words may be composed.

The observation map is part of the claim. Preserving a categorical answer is weaker than preserving all output probabilities, and preserving either can be weaker than preserving future behavior after another operation. The companion Neuralese manuscript develops that distinction in detail. An OSL claim must not silently change its observation interface after evaluation.

2.2 Operational packets are not geometric projectors

An OSL packet is a record with content and metadata. A Komorebi geometric packet is a component of a decomposition, often represented by a projector or a frame. These concepts may be linked by an application, but they are not definitionally identical.

A useful abstract operational packet is

$$p = (\mathrm{id}, \mathrm{content}, \mathrm{origin}, \mathrm{access}, \mathrm{status}, \mathrm{scope}, \mathrm{evidence}).$$

The implementation has a more detailed schema. The content may describe a vector or an operation, but it may also be an ordinary proposition. The provenance records how the content entered the system. An access relation states which agents may inspect it. Endorsement and authorization are separate relations. A packet's numerical confidence does not itself grant either.

2.3 Meaning is relative to a declared task family

Two programs can have the same output on today's input and still differ on another input, on their receipts, or on their permitted effects. We therefore distinguish state equivalence, observation equivalence, contextual equivalence, and trace equivalence. A discovery algorithm that quotients programs by their final finite-state transformation is not thereby entitled to erase provenance or proof consumption from a security-sensitive trace. The intended quotient must be named.

3 Operator transport and compositional semantics

3.1 Exact transport

Let $A_g : X \to X$, $B_g : Y \to Y$, and $T : X \to Y$ be linear maps. We use column vectors and execute the rightmost map first. The compatibility equation is

$$B_g T = T A_g. \tag{1}$$

When this equation holds, translating a state and applying the target operator agrees with applying the source operator and then translating. It is a statement about operations, not merely geometric resemblance between embeddings.

Proposition 3.1 (Compositional transport)**.** *If* (1) *holds for every generator, then for every finite word* $w = g_1 \cdots g_k$*, with* $A_w = A_{g_k} \cdots A_{g_1}$ *and analogously for* B_w*, one has* $B_w T = T A_w$*.*

Proof. The empty word gives $T = T$. If the identity holds for w, then $B_g B_w T = B_g T A_w = T A_g A_w$. Induction on word length proves the result. □

No invertibility of T is needed. When T is invertible, the target operators are conjugates $B_g = T A_g T^{-1}$. When it is not, the equation may describe an abstraction or information-losing projection. That projection must preserve the observations relevant to the advertised application; the intertwining equation alone does not specify which observations matter.

3.2 Approximate transport

Proposition 3.2 (Finite-word error budget). *Suppose $E_i = B_{g_i} T - T A_{g_i}$, with $\|E_i\| \le \varepsilon_i$. Then*

$$B_w T - T A_w = \sum_{i=1}^{k} B_{g_k} \cdots B_{g_{i+1}} E_i A_{g_{i-1}} \cdots A_{g_1}.$$

Consequently, if every $\|A_g\| \le a$ and $\|B_g\| \le b$,

$$\|B_w T - T A_w\| \le \sum_{i=1}^{k} b^{k-i} \varepsilon_i a^{i-1}.$$

Proof. Insert the intermediate products obtained by moving T past one operator at a time. Consecutive products cancel, leaving the displayed sum. Submultiplicativity and the triangle inequality give the norm bound. □

For $a = b = 1$ and $\varepsilon_i \le \varepsilon$, the bound is $k\varepsilon$, not ε. A small one-step residual does not establish unlimited reliable composition. If the operators amplify errors, the budget can grow faster. Practical certificates must record their admissible operation family, norms, and composition horizon rather than advertising an unqualified transport accuracy.

3.3 Projectors and changes of chart

For an invertible T, define $P' = TPT^{-1}$. If $P^2 = P$, then

$$(P')^2 = TP^2T^{-1} = P'.$$

Idempotence is preserved by similarity. Orthogonality with respect to the Euclidean inner product need not be. If a metric G is transported to $G' = T^{-\mathsf{T}} G T^{-1}$, metric-relative identities can be compared consistently. These elementary distinctions matter in a language that allows an operator to be called a projector. A name or an approximate matrix test must not conceal the metric or tolerance under which the predicate is evaluated.

The exact derivations in this section are standard linear algebra. Their role is to supply explicit contracts for OSL, not to introduce a new theory of intertwiners.

4 The operational state machine

4.1 State and judgment forms

A schematic runtime state is

$$S = (\Gamma, V, \mathcal{O}, \mathcal{P}, \mathcal{Q}, \mathcal{M}, \mathcal{C}, \mathcal{A}, \mathcal{T}),$$

where Γ contains declarations, V vectors, $\mathcal{O}$ operators, $\mathcal{P}$ packets, $\mathcal{Q}$ proof resources, $\mathcal{M}$ memories, $\mathcal{C}$ capabilities, $\mathcal{A}$ planned actions, and $\mathcal{T}$ contracts, sessions, and transaction state. The concrete runtime also maintains metadata and audit structures. This tuple is an explanatory grouping, not a replacement wire schema.

A typing judgment $\Gamma \vdash e : \tau$ constrains an expression's carrier or kind. A proof-resource judgment constrains which already established obligations may be consumed. An execution judgment

$$\langle c, S, L \rangle \longrightarrow \langle S', L' \rangle$$

relates a source statement, operational state, and audit ledger to their successors. A statement that rejects still has an outcome and, within the implemented lifecycle, an auditable terminal explanation. Rejection is not equivalent to absence of a trace.

4.2 Access is not ownership

A received observation is evidence that someone else observed something, not a firsthand observation by its recipient. The implemented language preserves the distinction between observed, communicated, inferred, simulated, injected, and reconstructed content. [O03, O05]

Proposition 4.1 (Provenance preservation for a communication rule)**.** *Suppose a packet's origin record is immutable under a communication operation, and communication adds a recipient access record plus a communicated provenance step. Any finite sequence of such operations preserves the original origin record. No recipient acquires a firsthand observation solely by that sequence.*

Proof. Each permitted step leaves the origin component unchanged. Induction preserves that component over the entire sequence. The only new event constructor is communication; none of the steps introduces a firsthand-observation event for a recipient. □

This is a theorem about the stated rule, not an external guarantee that a source really observed the world. A false input observation can be faithfully propagated. OSL separates that world-grounding problem from the internal consistency of provenance.

4.3 Proof resources and permissions

Persistent, affine, and linear proof resources have different use disciplines. A persistent resource may be reused, an affine resource used at most once, and a linear resource used exactly once in the relevant completed scope. These are operational restrictions on evidence use, not statements that truth itself expires.

Proposition 4.2 (Finite linear-use accounting)**.** *Assume a linear resource starts with use count one, every legal consumption requires a positive count and decrements it by one, and successful scope completion requires count zero. Then a successful scope consumes the resource exactly once.*

Proof. A second consumption is impossible because the first makes the count zero. Zero consumptions leave count one and prevent successful completion. Exactly one consumption is therefore necessary and sufficient for this accounting condition. □

A capability is a different object. It grants a bounded kind of action to a named agent, optionally with a use limit and deadline. A proof about content cannot substitute for the capability required to execute an effect. In this reference implementation effects and capabilities are symbolic. They do not impose an operating-system sandbox or authorize a real external tool.

4.4 Rollback and evidence

Separate mutable operational state from an append-only audit ledger. A rollback restores the selected operational snapshot and appends a rollback event, but does not restore the ledger to its old prefix. Otherwise a failed speculative action could erase the evidence that it was attempted.

Proposition 4.3 (Rollback does not imply historical erasure)**.** *Under the preceding transition rule, restoring an operational snapshot preserves all earlier audit entries and appends the rollback event. Operational equality with an earlier state therefore does not imply equality of the two execution histories.*

Proof. The post-rollback operational component equals the saved snapshot by definition. The ledger is extended rather than replaced. The new ledger consequently contains a strict extension of the earlier prefix whenever a rollback event is appended. □

This does not undo physical effects. A program that has already acted on an external environment requires an application-level compensation or recovery protocol. The reference language's rollback concerns its declared internal projection.

5 Receipt integrity and independent checking

5.1 Four assurance questions

For a receipt-bearing execution, ask four separate questions. Are the bytes intact? Does the trace bind the exact source? Are the recorded transitions semantically valid within a checked domain? Does executing the implementation again reproduce the result? Core 0.5 exposes separate answers rather than counting same-runtime replay as independent semantic evidence. [O05]

A digest commits to a particular source or record, conditional on the hashing assumptions. It does not establish the source's truth, authenticate a principal, or prove that a claimed transition occurred. The semantic checker must connect operands, prior state, effects, and final projections. An externally retained chain tip is needed to detect removal of a suffix from an otherwise valid unanchored history.

5.2 The current checker stack

The components below answer different assurance questions. A record may pass one layer and still require separate evidence in another.

Component	Checked subject	Remaining dependency
Receipt-hardened base	Source lifecycle, packet and proof state, capabilities, actions, protected rollback	Primary parser and static checker remain shared.
Numeric component	Spaces, dimensions, matrices, applications, transports, numerical predicates	Finite floating-point profile and supported grammar.
Contract component	Five implemented templates, obligations, deadlines, final checks	Not arbitrary temporal logic.
Session component	Finite graph checks and synchronous state/history transitions	Not asynchronous distributed liveness.
Metadata component	Secret commitments, records, assertion counts, show bindings	Does not prove all assertion predicates.
Intrinsic component	The exact prior prefix checked by `verify_receipts`	Chain integrity, not unrelated semantic truth.
Supplementary replay	Receipt and final-state reproduction	Same interpreter family, not an independent semantics.

The numerical, contract, and session fragments have separate bounded parsers. The metadata and intrinsic paths share more of the primary frontend. "Independent" is therefore a scoped implementation property, not a claim that every component has a disjoint trusted computing base.

5.3 Unclosed runtime obligations

The preserved Core 0.5 specification explicitly leaves several obligations open: the general truth of all fourteen assertion forms; exact reconstruction of every complete cross-domain transaction digest; independent reconstruction of every rejection cause; exact diagnostic event-count reconstruction; and removal of all shared frontend dependencies. Python, NumPy, canonical serialization, hashing, and the host remain trusted. These are not silently discharged by this edition. [O05]

A sound production claim would require a stronger semantic correspondence theorem or an equivalently explicit assurance argument. The finite tests establish that the tested implementation handles the included cases, not that every syntactically valid program satisfies all intended invariants. The older 47,958-transition metatheory concerns declared finite abstract models, not an exhaustive traversal of the Python interpreter's state space. [O03]

6 Bounded discovery and semantic admission

The Discovery Bridge supplies a missing distinction between *a well-formed proposal* and *an identified operator*. It accepts a complete finite world W, a primitive vocabulary G_W, a depth bound d, demonstrations D, and explicitly claimed laws. All of these belong to the certificate's scope. [ODB]

6.1 Transformation classes

Let S_W be the full finite legal state set and

$$\mathcal{P}_{W,d} = \bigcup_{j=0}^{d} G_W^j, \qquad \sigma_W(p) = \big(p(s)\big)_{s \in S_W}.$$

Programs are identified when their complete state signatures agree. For a demonstration set $D = \{(s_i, t_i)\}$, define

$$\mathcal{C}(D) = \{[p] : p \in \mathcal{P}_{W,d},\ p(s_i) = t_i \text{ for all } i\}.$$

The implementation enumerates this bounded universe, quotients equivalent spellings, and tests the advertised laws on every state in S_W. It admits only a unique derived transformation that is neither the identity nor an existing primitive and whose listed laws pass.

Proposition 6.1 (Scope-relative identification)**.** *Complete enumeration and exact evaluation identify one transformation precisely when $|\mathcal{C}(D)| = 1$. If $|\mathcal{C}(D)| > 1$, two compatible classes have a separating state in S_W. Adding the intended output at that state removes at least one of those classes.*

Proof. Distinct equivalence classes have unequal full signatures. At least one legal input therefore yields unequal outputs. A demonstration choosing one output is inconsistent with a class yielding the other. The uniqueness assertion follows directly from the definition of $\mathcal{C}(D)$. □

The proposition says nothing about programs outside the depth bound or operators outside the vocabulary. "New" means new relative to the declared primitive set. It is not a claim of mathematical novelty or autonomous concept invention.

6.2 The five possible outcomes

Outcome	Meaning
`ADMITTED`	One derived transformation fits and every listed law holds.
`AMBIGUOUS`	Several transformations fit; retain alternatives and a separating input.
`UNSATISFIABLE`	No candidate fits, or demonstrations conflict.
`REJECTED_FALSE_LAW`	The identified transformation violates an advertised relation.
`REJECTED_NONNOVEL`	The transformation is equivalent to identity or a primitive.

Ambiguity is an informative result, not a failed attempt to force admission. The bridge's verifier recomputes the candidate decision and the registry replays the embedded proposal. Merely supplying a content digest cannot bypass that semantic reconstruction.

6.3 Why two equivalent programs can still require separate traces

In the four-state flip example, `flip_left; flip_right` and `flip_right; flip_left` have the same full state transformation. A transformation library should identify them. An audit ledger may nevertheless distinguish their execution order, resource use, or intermediate observations. Quotienting the first interface does not authorize quotienting the second. A typed bridge must state which equivalence it certifies.

7 Worked OSL programs

The following programs are taken from the recovered finite corpus. They are executable reference syntax, not pseudocode. The complete sources and their fresh receipts are bundled. [O05]

7.1 A projector transported between coordinates

```
module coordinate_transport
space H dim 2
operator P : H -> H kind projector matrix [[1,0],[0,0]]
operator T : H -> H kind linear matrix [[0,-1],[1,0]]
transport P via T as P_rot
check_projector P_rot tolerance 1e-12
vector x in H values [0.0,3.0]
apply y = P_rot(x)
assert vector y approx [0.0,3.0] tolerance 1e-12
```

Here P selects the first coordinate and T rotates the chart. The transported projector selects the second coordinate. The program checks the transported predicate and applies it to a vector in that component. It does not infer a correspondence between two learned models; the transport matrix is supplied.

7.2 Communication without fabricated firsthand memory

The complete communication example declares separate lineages for A and B, records an observation by B, communicates it to A, and permits A to use it after the required endorsements and authorization. Its concluding assertions include

```
assert packet route_blocked_at_A provenance communicated
assert memory A contains route_blocked_at_A as communicated true
assert memory A contains route_blocked_at_A as observed false
```

The successful action and the origin of its supporting evidence are distinct claims. A recipient may act on communicated evidence without relabeling it as its own observation.

7.3 From quarantine to admission

```
module neural_candidate_admission
lineage L_A
agent A lineage L_A
neural_propose latent_claim to A feature feature_441 content "route-may-be-blocked" confidence 0.84
assert packet latent_claim status provisional
assert packet latent_claim obligation origin true
attest origin latent_claim by A lineage L_A as p_origin
attest relevance latent_claim by A for planning as p_relevance
admit latent_claim by A using ["p_origin","p_relevance"]
assert packet latent_claim status admitted
assert packet latent_claim provenance inferred
assert packet latent_claim quarantined_by A false
```

This example also exposes a trust boundary. The attestations are declared inputs to the language; they are not generated by a universal truth detector. A deployment that requires semantic validation must bind the attester to an actual external checking procedure. The Discovery Bridge and the new finite

interface demonstrate two narrowly specified checking procedures. Neither establishes truth outside its declared world.

8 A checked semantic-checkpoint interface

8.1 What the new demonstrator does

The edition adds `interface/semantic_interface.py`. It takes a *fully declared finite deterministic machine*, discovers its observable quotient, checks a certificate through local preservation equations, and serializes a checkpoint containing the quotient class and its scope bindings. It is not a decoder of arbitrary Neuralese vectors. Its entire transition table is an explicit trust root.

Let the declared machine be $M = (S, \mathcal{A}, F, O)$. The certificate contains a surjection $q : S \to Q$, quotient transitions $\bar{F}_a$, and outputs $\bar{O}$, subject to

$$q(F_a(s)) = \bar{F}_a(q(s)), \qquad O(s) = \bar{O}(q(s)) \quad (s \in S,\ a \in \mathcal{A}). \tag{2}$$

The certificate generator uses partition refinement. The checker instead verifies the displayed equations over every declared state and action. It accepts any sound quotient, including a finer nonminimal one. Minimality is a property of the generator and its tests, not an extra property falsely attributed to the certificate checker.

Theorem 8.1 (Finite semantic checkpoint preservation)**.** *If* (2) *holds, then a checkpoint storing $q(s)$ reproduces $O(F_w(s))$ for every finite word w over the declared alphabet, when continued with the quotient transitions. No information inside a fiber of q is needed for those outputs.*

Proof. Induction on $|w|$ gives $q(F_w(s)) = \bar{F}_w(q(s))$. Apply the output equation to the terminal state. The argument uses totality of the declared transition maps and the equations at every state, not merely a sampled history panel. □

This finite theorem permits all word lengths *for the declared finite model*. It is not an infinite-horizon theorem about a learned implementation that only approximately matches that model.

8.2 The D4 fixture

The demonstrator has sixteen concrete states (a, b, u), with $a \in \mathbb{Z}/4\mathbb{Z}$, $b, u \in \mathbb{Z}/2\mathbb{Z}$. Its visible state is (a, b); u is a private bit. Operations are

$$r(a, b, u) = (a + 1, b, u), \quad f(a, b, u) = (-a, 1 - b, u), \quad v(a, b, u) = (a, b, 1 - u).$$

The first two satisfy $r^4 = f^2 = frfr = 1$. The private operation changes only u. Partition refinement recovers eight semantic classes. Discarding the private bit preserves every future declared observation.

The source-bound certificate is then rendered into OSL. A validated finite checkpoint is admitted under the adapter's explicit attestation policy. An unverified candidate remains provisional and quarantined. Both paths execute in Core 0.5 and pass its base and semantic checkers. The accepted path has 27 receipts and the unverified path 23. This is an actual cross-package execution, but its producer is a finite fixture rather than a trained model.

8.3 Binding, restoration, and limits

A checkpoint binds the machine digest, certificate digest, scope, quotient class, and checkpoint digest. A fresh process restores it only against the same declared machine and a valid quotient certificate. Changing an alphabet, output table, or scope invalidates the old binding. A valid serialized copy establishes informational continuation, not personal identity or authenticated process lineage.

A rehashed checkpoint that names a different valid class is still a different valid checkpoint. Without an external authenticating authority, the checker cannot determine which past process truly produced it. Content validity, historical authenticity, and authorization remain separate.

9 What the retained neural-language experiments establish

The OSL archive includes geometry, micro-emergence, Hugging Face causal, and operand-slot experiments. They are preserved because the language program depends on understanding what neural producers can and cannot supply. They do not change the identity of the canonical runtime. [OL]

The geometry pilot recovered partial structure under added supervision and equivalence pressure, but control models weakened the inference that a usable internal operator algebra had emerged. The Discovery Bridge's upstream audit also identified a problem in an older intervention endpoint: the experimental definition of `observe(a,f)` changed both status and origin, while the intervention scripts demanded preservation of the previous communicated origin. The resulting low conversion statistic is not a clean estimate of the intended observe operation. It must not be used as a definitive numerical falsification of that operation. The valid conclusion is that the desired causal-language claim was not established by the retained experiment. [ODB]

The operand-slot v0.3.1 package supplies entities, operands, source reads, and write locations, but no completed scientific run. Its supplied ontology would make a future success a test of scaffolded execution, not unrestricted discovery. One audit also leaves alpha-isomorphic separation unverified. These qualifications are necessary because a representation can predict labels without implementing the intended updates.

The practical interface is therefore asymmetric: neural computation may propose an operator or a state interpretation; the explicit language does not grant it authoritative semantics merely because a probe can decode it. Causal abstraction research motivates intervention-based tests, and critiques of unrestricted alignment motivate bounds on the complexity and information access of the translator. [G+23, S+25]

10 Fresh verification and reproducibility

10.1 Completed checks

Check	Result	Scope
Core 0.5 unit suite	168 passed	Packaged implementation tests.
Core 0.5 corpus	66/66	42 accepts, 24 expected rejections.
Native corpus receipts	1,825	Byte-identical regenerated bundle.
Discovery Bridge tests	20 passed	Core integration enabled.
Bridge programs in Core 0.5	3/3	71 receipts.
New finite-interface tests	15 passed	Includes exact finite enumeration and fresh-process restoration.
New OSL interface examples	2/2	Accepted and quarantined paths, 50 receipts total.

The conformance report, semantic summary, and complete source-bound trace bundle regenerate byte for byte from the preserved Core 0.5 source. The finite-interface tests include all 216 three-state, single-action machines with binary observations. This is a complete finite family, not a sampled machine-learning population. Larger neural experiment counts elsewhere in the repository are historical and are not included in this edition's fresh test total.

10.2 Reproduction commands

Run from the combined release root, with Python and the supplied dependencies installed:

```
python verify_release.py
python run_current_checks.py
python interface/run_demo.py --output ../local_demo
```

The first command checks the edition manifest and the absence of redistributed font files. The second runs the bounded current-code checks and writes separate logs. It does not train neural checkpoints or consume sealed learned-evaluation banks. Exact upstream commands and the original manifests remain in their own packages.

The reproducibility claim has limits. Test execution uses the installed Python and numerical stack documented in the validation record. It does not reproduce a historical GPU serving environment. Runtime matching of the finite OSL corpus is stronger here than a general numerical-portability assertion: the named files are actually identical, whereas no such claim is made for all neural floating-point trajectories.

10.3 A corrected invocation failure

The first bridge corpus command supplied a relative examples path. The upstream generator expected a path under its absolute root and raised `ValueError` before evaluating a program. Repeating the command with the absolute path completed all three cases. The original source was not changed. This invocation failure and the successful correction are preserved in the execution notes rather than hidden behind the last exit code of a compound shell command. A separate integrity invocation on the shortened directory name `core05` failed the upstream archive-name predicate, even though its payload checks passed. Repeating it on a canonical-named working copy passed all 156 listed file checks. The portable edition runner preserves that original root name in its temporary copy; it does not weaken the upstream

check.

11 Compatibility, trust, and publication status

OSL is useful because its semantics are explicit enough to disagree with a producer. A checker may reject a proposal, retain ambiguity, or accept only a restricted scope. A language that simply translates every fluent claim into an authoritative memory would lose the point of this design.

There are two distinct remaining frontiers. The *runtime frontier* is a mechanized correspondence between all implemented source forms, their effects, and an independently specified semantics. The *learning frontier* is a neural producer that discovers reusable typed operations under controlled intervention and transfer tests. Completing one does not complete the other. A language can be useful before it is learned spontaneously; a learned latent operator can be causally meaningful before it has a faithful textual rendering.

The final claim of this edition is therefore direct: OSL is a specified and executable operator-symbolic research language with independently checked finite semantic components, exact bounded operator admission, and an implemented finite checkpoint bridge. The record does not establish a universal verifier or an endogenous neural language. Those are research targets, not missing editorial labels.

A Claim register

ID	Statement	Status and boundary
O1	Exact generator transport composes.	Proved by induction; specified maps and words.
O2	Approximate transport has a finite-word error budget.	Proved by telescoping; norm assumptions explicit.
O3	Communication preserves the original origin record.	Proved for the declared rule; external truth not implied.
O4	Linear-use accounting permits exactly one consumption in a completed scope.	Proved under the accounting rules.
O5	Rollback need not erase execution history.	Proved for separated state and append-only ledger.
O6	Finite demonstrations identify an operator only relative to a candidate universe.	Proved; implemented by Discovery Bridge.
O7	A locally checked finite quotient preserves all declared future outputs.	Proved; implemented and tested.
O8	Core 0.5 passes the retained corpus and 168-test suite.	Fresh execution in this edition.
O9	The complete Python interpreter has a mechanized soundness proof.	Not established.
O10	The neural producer has learned OSL's semantics generally.	Not established.
O11	The finite interface compiles arbitrary learned Neuralese.	Not claimed; it uses complete declared tables.

B Source hierarchy and reading map

The authority order is source-specific, not a total order by version. Current finite runtime semantics come from `OSL/core05/`. The historical broad mathematical development is preserved in `OSL/historical_monograph/`. Bounded synthesis is in `OSL/discovery_bridge/`. The newly authored finite checkpoint code is in `interface/`. The whole OSL lineage archive retains the experimental packages and earlier receipt-hardening release.

The appendix does not replace the original 88-page manuscript. That manuscript remains an independently identifiable historical artifact. Its statements about Core 0.3 describe that version, not Core 0.5. The new edition provides the missing version crosswalk and current evidence rather than silently rewriting an old release's dates and claims.

For a syntax-first reading, start with the current examples and `CORE05_SEMANTIC_INDEPENDENCE_SPEC.md`. For finite discovery, read `FORMAL_NOTES.md` before running the three proposal examples. For neural producers, read the companion Neuralese edition and the individual experimental claim boundaries. A reader should be able to trace a claim to its implementation and retained result without inferring authority from a filename containing "final."

C A compact operational checklist

Before accepting an externally produced representation, specify its carrier, observation interface, allowable future operations, history or state domain, source/version binding, and intended use. Check the obligations that actually belong to that scope. Preserve unresolved alternatives and failed conditions. Record whether the evidence is exact finite enumeration, a mathematical theorem, a sampled experiment, or a declaration by a trusted authority.

Before executing an action, separately verify the required capability and resource state. Before compressing memory, verify that the proposed abstraction preserves the future observations the consumer needs. Before translating between agents, distinguish a coordinate change from an information-losing abstraction. Before declaring a learned language, establish causal intervention and continuation behavior rather than relying only on decoded labels.

These are operational consequences of the formal distinctions in this paper. They are not claims that a single finite reference language settles institutional governance, world truth, or general AI safety.

References

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *POPL*, 238–252, 1977. Author-maintained bibliographic record: https://www.di.ens.fr/~cousot/COUSOTpapers/POPL77.shtml.

[N97] George C. Necula. Proof-carrying code. *POPL*, 106–119, 1997. DOI: 10.1145/263699.263712.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *POPL*, 273–284, 2008. DOI: 10.1145/1328438.1328472.

[G+23] Atticus Geiger et al. Causal Abstraction: A Theoretical Foundation for Mechanistic Interpretability. *arXiv:2301.04709*, initially 2023, versioned research manuscript. https://arxiv.org/abs/2301.04709.

[H+24] Shibo Hao, Sainbayar Sukhbaatar, DiJia Su, Xian Li, Zhiting Hu, Jason Weston, and Yuandong Tian. Training Large Language Models to Reason in a Continuous Latent Space. *arXiv:2412.06769*, initially 2024. https://arxiv.org/abs/2412.06769. Cited for the latent-recurrence proposal, not as a result obtained in this release.

[S+25] Denis Sutter, Julian Minder, Thomas Hofmann, and Tiago Pimentel. The Non-Linear Representation Dilemma: Is Causal Abstraction Enough for Mechanistic Interpretability? *arXiv:2507.08802*, 2025. https://arxiv.org/abs/2507.08802.

[O03] Komorebi P. *Operator-Symbolic Languages for Artificial Intelligence: Finite Metatheory and Independent-Kernel Draft v0.5*. OSL-Core 0.3, 2026. Preserved manuscript, source, finite models, and verification records in `OSL/historical_monograph/`.

[O05] Komorebi P. *OSL-Core 0.5 Semantic Independence v0.1.0*. 2026. Current runtime, specifications, corpus, tests, and native receipts in `OSL/core05/`; fresh replay in `OSL/validation/`.

[ODB] Komorebi P. *OSL Discovery Bridge v0.1.0*. 2026. Finite world semantics, admission algorithm, upstream audit, and tests in `OSL/discovery_bridge/`.

[OL] Komorebi P. *OSL Complete Lineage with Core 0.4*, 1 September 2026. Includes geometry, micro-emergence, causal-compositional, operand-slot, and receipt-hardening branches. Exact original bytes and hashes are bound by the edition's source ledger.

[FI] Komorebi P. *Finite Semantic Checkpoint Interface*, research edition 1.0, 2026. Newly authored reference demonstrator, tests, and actual OSL admission/quarantine traces in `interface/`.