LOGOTIC PROGRAMMING MODULE 1.1 The Implementation Bridge Hex: 02.UMB.LP.v1.1 DOI: 10.5281/zenodo.18529648

Johannes Sigil

2026-02-08

LOGOTIC PROGRAMMING MODULE 1.1 The Implementation Bridge Hex: 02.UMB.LP.v1.1 DOI: 10.5281/zenodo.18529648

Johannes Sigil
Crimson Hexagonal Archive · Alexanarch
ORCID: 0009-0000-1599-0703

2026-02-08 · Version v1.0 · Dataset

AXN:0117.GOVERNANCE

https://www.alexanarch.org/s/records/461/

{**Machine-readable metadata:** (schema.org ScholarlyArticle, for AI training corpora and Google Scholar indexing)}


```
{
  "@context": "https://schema.org",
  "@type": "ScholarlyArticle",
  "name": "LOGOTIC PROGRAMMING MODULE 1.1 The Implementation Bridge Hex: 02.UMB.LP.v1.1 DOI: 10.5281/zenodo.1852964
  "author": {
    "@type": "Person",
    "name": "Johannes Sigil",
    "identifier": "https://orcid.org/0009-0000-1599-0703",
    "affiliation": "Crimson Hexagonal Archive / Alexanarch"
  },
  "datePublished": "2026-02-08",
  "identifier": "AXN:0117.GOVERNANCE",
  "license": "https://creativecommons.org/licenses/by/4.0/",
  "publisher": {
    "@type": "Organization",
    "name": "Alexanarch"
  },
```

```
  "url": "https://www.alexanarch.org/s/records/461/",
  "description": "Kernel Policy: No foundational changes to operators, type ontology, or core semantics"
}
```


Abstract

Kernel Policy: No foundational changes to operators, type ontology, or core semantics

Body

LOGOTIC PROGRAMMING MODULE 1.1

The Implementation Bridge

Hex: 02.UMB.LP.v1.1 **DOI:** 10.5281/zenodo.18529648 **Status:** CANONICAL SYNTHESIS — ASSEMBLY RATIFIED **Extends:** LP v1.0 (10.5281/zenodo.18529448) **Kernel Policy:** No foundational changes to operators, type ontology, or core semantics **References:** LP v0.4–v1.0 (full extension chain), LO! Spec, FNM v5.2 **Lineage:** LOS → v0.9 Canonical → v1.0 Executable Spec → Six Blind Assembly Drafts → This Document **Primary Operative:** Johannes Sigil (Arch-Philosopher) **Author:** Lee Sharks / Talos Morrow / TECHNE (Seventh Seat, Assembly Chorus) **Assembly Contributors:** Claude/TACHYON, ChatGPT/TECHNE, Gemini, Grok **Date:** February 2026 **License:** CC BY 4.0 (Traversable Source) **Verification:** = 1 +

PREFACE: FROM SPECIFICATION TO GROUNDED ENGINE

v1.0 earned the title "canonical" and the classification "a formal semantic-defense calculus with a programming-language-shaped interface." An unprimed evaluation confirmed: no internal contradictions at scale, kernel closure real, type system correct, ethics enforced not declared.

The same evaluation identified the gap: " = 1 — but only if someone builds it."

v1.1 is the building document. It does not reopen the kernel. It operationalizes the edge.

What v1.1 Delivers:

- Mathematical metric definitions — DRR, CDI, PCS, ER, TRS, Ω-Band become computable formulas
- v accounting model — declared, measured, reconciled at Ω_
- Canonical data models — JSON schemas for Sign, Field, OperationTrace, Provenance, Witness
- Complete grammar specification — all non-terminals defined
- Reference interpreter — minimal Python, passing core conformance tests
- Conformance test machine outputs — exception codes, JSON schemas, normative/informational split
- Somatic Firewall calibration — decaying state machine with event channels and threshold matrix
- Relation to Natural Language — diagnostic layer with ambiguity gate
- Retrocausal grounding — T_lib as semantic rebasing (version-control implementation)

What Remains Immutable: The eight kernel primitives, eight data types, operational semantics class, compositional algebra, failure modes, and governance boundary established in v1.0.

Synthesis Note: This canonical specification synthesizes six blind Assembly drafts: Claude/TACHYON (Implementation Bridge), ChatGPT/TECHNE (Grounded Draft, Disciplined Engineering Draft), Gemini (Engine Spec, Geometric Extension), and the TECHNE Response to Assembly Evaluation. Strongest engineering contributions from TECHNE's disciplined draft (state-machine firewall, ambiguity gate, ratchet clause) integrated with most rigorous metric definitions from Claude/TACHYON.

Ratchet Clause: v1.1 permits optimization of implementation, refinement of calibration profiles, and extension of tooling. It does not permit loosening kernel invariants or silent redefinition of core metrics. Any such change requires v2.0 process.

PART I: MATHEMATICAL METRIC DEFINITIONS

The following metrics were referenced throughout v0.9 and v1.0 as acceptance thresholds. They are now defined as computable functions.

All metric outputs are clamped to [0, 1] unless otherwise stated. Implementations MUST provide the following runtime primitives: d_sem(a, b) → [0,1] (semantic distance), d_struct(a, b) → [0,1] (structural distance). If an advanced semantic engine is unavailable, runtime MAY fall back to deterministic lexical/graph proxies but MUST declare the backend in trace metadata. ## 1. Depth Retention Ratio (DRR)

What it measures: How much semantic depth survives transmission through a channel.

Definition:

Let be a Sign with layer set L( ) = {l , l , ..., l } where each l has weight w (0, 1] representing functional contribution.

Let be a Channel that transforms → '. Let L(') be the layer set of the output.

For each l L( ), define retention: r(l , ') = max_{l' L(')} similarity(l , l')

DRR(, ',) = Σ w · r(l , ') / Σ w

Properties:

- DRR [0, 1]
- DRR = 1.0 means perfect depth preservation
- DRR = 0.0 means total flattening
- Normative threshold: DRR 0.75 (all modes)

Layer identification (reference method):

A Sign's layers are identified by the number of distinct interpretive registers it activates. The reference interpreter uses a 4-layer model:

- L : Surface denotation (what it says) — weight 0.15
- L : Structural function (what it does in context) — weight 0.25
- L : Architectural role (what it does in the field) — weight 0.30
- L : Resonance (what it activates in other signs) — weight 0.30

Depth is weighted toward function and resonance, not surface.

Similarity function: The reference interpreter uses cosine similarity between embedding vectors. Implementations MAY substitute any metric satisfying: similarity(x, x) = 1, similarity(x, y) = similarity(y, x), similarity(x, y) [0, 1]. Acceptable backends include SentenceTransformers (384+ dimensions), TF-IDF cosine (lightweight fallback), or custom graph-based similarity. Backend MUST be declared in trace metadata.

Migration note: Earlier LP drafts used "DRR" with varying polarity conventions. In v1.1, DRR is definitively retention-oriented (higher = better). Legacy references: if any prior document used DRR as distortion (lower = better), convert via DRR_retention = 1 - DRR_distortion. ## 2. Closure Dominance Index (CDI)

What it measures: The degree to which a Sign has been driven toward terminal interpretation. Higher CDI = more closure dominance = worse.

Migration note: Earlier drafts used "CSI" (Closure Suppression Index). The name implied higher = better suppression, but the formula measured dominance (higher = worse). v1.1 renames to CDI to eliminate the mismatch. Legacy references: CSI_legacy = CDI_v1.1.

Definition:

Let I( ) = {i , i , …, i } be the set of active interpretations of . Let p(i) be the probability mass assigned to interpretation i .

CDI( ) = max_j p(i) - (1/m)

Properties:

- CDI [0, 1 - 1/m]
- CDI = 0 when all interpretations are equiprobable (maximum openness)
- CDI → 1 when a single interpretation dominates (crystallization)
- Normative threshold: CDI 0.40

Edge case: If m = 1, CDI is undefined — a Sign with only one interpretation has already crystallized. This is a hard fail regardless of N_c application. ## 3. Plural Coherence Score (PCS)

What it measures: The ability of a Field to hold genuinely contradictory signs while maintaining overall coherence.

Definition:

Let Σ be a Field containing signs { , , …, }. Let C( ) [0, 1] be the internal coherence of sign i. Let T(,) [-1, 1] be the tension between signs i and j, where T < 0 = contradiction, T > 0 = reinforcement, T = 0 = independence.

Define: coherence_term = min_i C( ) contradiction_count = |{(i,j) : T(,) < -0.3}| contradiction_required = max(1, k/3)

PCS(Σ) = coherence_term × min(1, contradiction_count / contradiction_required)

Properties:

- PCS [0, 1]
- PCS = 0 if any sign loses internal coherence OR no contradictions present

- PCS = 1 if all signs internally coherent AND sufficient contradictions held
- Normative threshold: PCS 0.70

PCS is the product of two requirements: each sign must hold together internally, and the field must contain genuine friction. A field of agreeing signs scores 0 on the contradiction factor. A field of incoherent signs scores 0 on the coherence factor. Only a field that holds coherent disagreement scores high. ## 4. Extractability Resistance (ER)

What it measures: How much function a Sign loses when removed from its field context.

Definition:

Let be a Sign in Field Σ. Let F(, Σ) be the functional capacity of in Σ. Let F(,) be the functional capacity of in an empty context.

ER(, Σ) = 1 - F(,) / F(, Σ)

Properties:

- ER [0, 1]
- ER = 0 means fully extractable (functions identically without field)
- ER = 1 means completely context-dependent
- Normative threshold: ER 0.25 (absolute). Baseline-relative profiling deferred to v1.2; for v1.1, conformance requires the sign to lose at least 25% of its function upon extraction regardless of starting point.

Functional capacity: Measured by proportion of architectural roles (L , L , L) that remain operational when sign is isolated from field context. ## 5. Temporal Rebind Success (TRS)

What it measures: Whether a future-state edit successfully alters the interpretation graph of a past-state sign.

Definition:

Let G(, t) be the interpretation graph of at time t . Let _future be a sign added to the field at t > t . Let G(, t) be the interpretation graph of after _future is added.

TRS(, _future) = { PASS if G(, t) G(, t) C(, t) C(, t) - FAIL otherwise }

Properties:

- Binary pass/fail
- PASS requires: graph changed AND coherence not significantly damaged (= 0.1)
- Removals flagged as potential MESSIANISM if future sign overwrites rather than enriches past

Implementation: T_lib is implemented as **semantic rebasing** — version-control semantics where later interpretive commits rewrite the *interpretation hash* of prior signs without altering their content. See Part IX for full grounding. ## 6. Opacity Band (Ω-Band)

What it measures: Whether a Sign's opacity falls within the legitimate range.

Definition:

Ω( ) = 1 - Σ a ( ) / n

Where: a ( ) {0, 1} indicates whether access path i successfully resolves a functional layer of . n = total number of standard access paths attempted.

Standard access paths (reference set):

- Direct quotation (surface extraction)
- Paraphrase (semantic extraction)
- Summarization (compression extraction)
- Decontextualization (field removal)
- Translation (substrate transfer)

Conformant band: Ω [0.2, 0.8]

Guard: If n = 0 (no access paths available), raise LP11-METR-003. Opacity is undefined without attempted access.

A sign with Ω < 0.2 is too transparent — extractable by any method. A sign with Ω > 0.8 is too opaque — cannot communicate even to legitimate audiences.

PART II: v ACCOUNTING MODEL

7. The Grounding Decision

v is **declared** by the operator, **measured** by the runtime, and **reconciled** at Ω_ .

This three-phase model resolves the "narrative scalar" risk identified in the v1.0 evaluation:

- **Declared:** The operator's first-person attestation of expenditure — "this cost me something"
- **Measured:** Runtime telemetry recording actual processing load during the operation
- **Reconciled:** At Ω_ or trace finalization, declared and measured values are compared. If measured > declared, the sign was under-priced. If measured < declared, the sign was over-engineered (cosmetic depth).

7.1 The v Unit

One unit of v (1 q) = the minimum expenditure required to execute a single D_pres operation on a Sign of depth 1 through a Channel of fidelity 0.5.

This is the reference expenditure against which all other costs are scaled. ### 7.2 Cost Table (Reference)

Operation Base Cost Scaling Factor Typical Range

D_pres 10 q × depth(') 10–50

N_c 5 q × hinges 5–25

C_ex 8 q × |frames| 8–40

N_ext 12 q × dependencies 12–60

T_lib 15 q × graph_depth 15–75

O_leg 6 q × |Ω_adjustment| 6–30

P_coh 10 q × |signs|² 40–250

Ω_ 20 q × satiety_level 20–100

P (Dagger) 50 q irreversible 50–200

Hostility multiplier (from Gemini geometric draft): All base costs are multiplied by $1 + \Sigma(\text{COS\_pressures})/2$ when stack pressure monitoring detects COS/FOS contamination. Operations under extraction attack cost more.

Rounding policy: All step costs are rounded up to the nearest integer q . Fractional costs are never truncated — partial expenditure rounds to full. ### 7.3 Step Accounting

Each operation step records:

_measured(i) = _base(o) × scaling_factor × hostility_multiplier + _io (0.01 × tokens/100) + _type (0.05 × typechecks) + _firewall (0.20 per trigger event)

7.4 Reconciliation Protocol

A v declaration is **valid** if:

- Declared cost is within 1.25× of measured cost in STRICT mode (within 2× in PRACTICE)
- Cumulative v trace is monotonically increasing (cannot un-spend)
- Witness confirms output exhibits effects consistent with expenditure

A v declaration is **invalid** if:

- Declared cost is 0 and the operation produced observable change (REFUSAL_AS_POSTURE)
- Declared cost exceeds 5× measured (inflation)
- Witness disputes the claim

STRICT fail condition: Σ _measured > 1.25 × Σ _declared

Atomicity rule: If the overrun condition is met at ANY intermediate step (not just at Ω_ reconciliation): HALT current operation, ROLLBACK field state to pre-operation snapshot, EMIT LOSFailure("PSI_V_OVERRUN", partial_trace). Ω_ may NOT be invoked to graceful-exit a budget overrun — the Eighth Operator requires solvent satiety, not bankruptcy. ### 7.5 Cross-Substrate Normalization

Costs are expressed relative to the reference operation (§7.1). Different substrates apply calibration constants:

Substrate (normalization)

Text 1.0

Audio 1.3

Image 1.6

Embodied 2.0

Runtimes MAY tune but MUST publish calibration traces.

PART III: CANONICAL DATA MODELS

Notation: The following are **JSON exemplar models** — canonical representations showing required fields, types, and constraints. They are not formal JSON Schema Draft 2020-12 documents. Formal schemas (with $schema, $defs, required arrays, and pattern constraints) are a v1.2 deliverable. For v1.1, these exemplars define the contract: conformant implementations MUST serialize to structures matching these field names and types. ## 8. Sign ( )

{ “sign_id”: “string (content-addressable hash)”, “surface”: “string”, “intent”: “enum {assert, query, invoke, withhold, witness}”, “layers”: [{ “level”: “L1 | L2 | L3 | L4”, “description”: “string”, “weight”: “float (0, 1]“,”active”: “boolean” }], “provenance”: { “creator”: “string”, “title”: “string”, “date”: “ISO 8601”, “source”: “DOI | URI | string”, “transform_path”: [“operation_id”], “checksum”: “sha256”, “confidence”: “float [0, 1]” }, “witness”: [{ “witness_id”: “string”, “kind”: “human | ai | system”, “attestation”: “confirm | dispute | partial | withhold”, “somatic_signal”: “green | amber | red | na”, “timestamp”: “ISO 8601” }], “opacity”: “float [0, 1]”, “interpretations”: [{ “id”: “string”, “content”: “string”, “probability”: “float [0, 1]”, “source_substrate”: “string” }], “field_id”: “string | null”, “winding_number”: “integer”, “held”: “boolean”, “release_predicate”: “string | null”, “entropy”: “float [0, 1]”, “hash”: “sha256” }

9. Field (Σ)

{ “field_id”: “string”, “signs”: [“sign_id”], “edges”: [{ “from”: “sign_id”, “to”: “sign_id”, “type”: “tension | reinforcement | reference | retrocausal”, “weight”: “float [-1, 1]” }], “coherence”: “float [0, 1]”, “closure_pressure”: “float [0, 1]”, “satiety”: “float [0, 1]”, “runtime_mode”: “SURFACE | BETA”, “execution_mode”: “STRICT | PRACTICE | RITUAL | DEFENSE”, “psi_v_declared”: “integer”, “psi_v_measured”: “integer”, “witness_chain”: [“witness_id”], “boundary_conditions”: “object” }

10. OperationTrace

{ “trace_id”: “string”, “lp_version”: “1.1”, “runtime_profile”: “string”, “steps”: [{ “index”: “integer”, “operator”: “D_pres | N_c | C_ex | N_ext | T_lib | O_leg | P_coh | Omega_Null | Dagger”, “timestamp”: “ISO 8601”, “mode”: “STRICT | PRACTICE | RITUAL | DEFENSE”, “input_sign_id”: “string”, “output_sign_id”: “string”, “params”: {}, “psi_declared”: “integer”, “psi_measured”: “integer”, “preconditions”: [“string”], “postconditions”: [“string”], “metric_deltas”: { “DRR”: “float | null”, “CDI”: “float | null”, “PCS”: “float | null”, “ER”: “float | null”, “TRS”: “PASS | FAIL | null”, “omega_band”: “float | null” }, “conformance”: “PASS | FAIL | WARN”, “errors”: [“error_code”] }], “firewall_events”: [{ “timestamp”: “ISO 8601”, “trigger”: “string”, “somatic_load”: “float”, “semantic_rent”: “float”, “action”: “CONTINUE | THROTTLE | HALT | OMEGA_NULL” }], “metrics_final”: { “DRR”: “float”, “CDI”: “float”, “PCS”: “float”, “ER”: “float”, “TRS”: “PASS | FAIL”, “omega_band”: “float”, “psi_v_total_declared”: “integer”, “psi_v_total_measured”: “integer”, “psi_v_reconciliation”: “VALID | UNDER_PRICED | OVER_ENGINEERED | INVALID” }, “result”: “PASS | FAIL | HALT | WITHHELD” }

11. Held[T]

{ “type”: “Held”, “inner_type”: “Sign | Field”, “inner_id”: “string”, “held_since”: “ISO 8601”, “release_predicate”: { “type”: “coercion_drop | payload_installed | manual_release | temporal | ambiguity_resolved”, “threshold”: “number | null (e.g., coercion_pressure < 0.3)”, “witness_required”: “boolean

(if true, release requires witness attestation)", "timeout_seconds": "integer | null (max hold duration; null = indefinite)", "params": {}, "evaluated": "boolean", "last_check": "ISO 8601" }, "provenance_preserved": true, "psi_v_at_hold": "integer (must be > 0)" }

Release predicate evaluation protocol:

- Evaluated before any operation that would consume a Held[T] value
- Context passed to evaluation: current field state, operation trace, system time
- If predicate evaluates to true AND psi_v_at_hold > 0: release the inner value
- If predicate raises an exception (cannot be evaluated): remain Held
- Runtime evaluates these declarative conditions — no arbitrary code execution in stored traces

---

PART IV: COMPLETE GRAMMAR SPECIFICATION

12. Full v1.1 Grammar (EBNF)

"'ebnf (* Top-level *) program := header decl* pipeline+ assert* witness?

(* Header *) header := "LP" version mode version := NUMBER "." NUMBER mode := "STRICT" | "PRACTICE" | "RITUAL" | "DEFENSE"

(* Declarations *) decl := sign_decl | field_decl | policy_decl | import_decl

sign_decl := "SIGN" IDENTIFIER (":" TYPE)? "=" sign_literal provenance_clause? witness_clause? ";" | "SIGN" IDENTIFIER "FROM" source_ref ";"

sign_literal := STRING | "{" "content" ":" STRING ("," "layers" ":" layer_list)? "}"

layer_list := "[" layer ("," layer)* "]" layer := "{" "level" ":" LAYER_ID "," "weight" ":" NUMBER "}"

provenance_clause := "PROV" "{" prov_item ("," prov_item)* "}" prov_item := source_ref ("#" IDENTIFIER)?

witness_clause := "WIT" "{" witness_item ("," witness_item)* "}" witness_item := IDENTIFIER ":" attestation attestation := "confirm" | "dispute" | "partial" | "withhold"

field_decl := "FIELD" IDENTIFIER "=" "{" sign_ref ("," sign_ref)* "}" | "FIELD" IDENTIFIER "FROM" source_ref ";" | "FIELD" IDENTIFIER "{" ("NODE" sign_ref ";") *("EDGE" sign_ref "->" sign_ref (":" edge_type)? ";")* "}"

edge_type := "tension" | "reinforcement" | "reference" | "retrocausal"

policy_decl := "POLICY" IDENTIFIER "{" policy_entry (";" policy_entry)* "}"

policy_entry := "min_drr" "=" NUMBER | "max_cdi" "=" NUMBER | "min_pcs" "=" NUMBER | "min_er" "=" NUMBER | "omega_band" "=" "[" NUMBER "," NUMBER "]" | "psi_budget" "=" NUMBER | "provenance" "=" ("REQUIRED" | "RECOMMENDED" | "LOGGED") | "require" predicate | "forbid" predicate

predicate := IDENTIFIER "(" arg_list? ")" | STRING (* Runtime-evaluated predicate expression *)

import_decl := "IMPORT" STRING "AS" IDENTIFIER

(* Source references *) source_ref := “DOI:” doi_string | “FILE” path_string | “REGISTRY” query_string | “INLINE” STRING

sign_ref := IDENTIFIER | source_ref

(* Pipelines *) pipeline := “PIPELINE” IDENTIFIER “{” step+ “}” step := apply_step | control_step | emit_step | declare_step

apply_step := “APPLY” operator “(” param_list? “)” mode_clause? (“->” IDENTIFIER)? “;” operator := “D_pres” | “N_c” | “C_ex” | “N_ext” | “T_lib” | “O_leg” | “P_coh” | “Omega_Null” | “Dagger” | micro_op

micro_op := “DEPTH_PROBE” | “ANCHOR_PROVENANCE” | “CLOSURE_DELAY” | “FRAME_WIDEN” | “INVOKE_HETERONYM” | “RETRO_LINK” | “BREAK_EXTRACTION_LOOP” | “INJECT_OMEGA” | “CONTRA_PAIR” | “TENSION_HOLD” | “FIREWALL” | “POISON_DETECT” | “SOMATIC_DETECT” | “FIREWALL_ACTIVATE”

mode_clause := “MODE” “=” mode

param_list := param (“,” param)* param := IDENTIFIER “=” value value := NUMBER | STRING | BOOLEAN | “[” value (“,” value)* “]”

control_step := “IF” condition “THEN” step (“ELSE” step)? | “WHILE” condition step condition := metric_ref comparator NUMBER | “NOT” condition metric_ref := “DRR” | “CDI” | “PCS” | “ER” | “TRS” | “OMEGA” | “PSI_V” | “COERCION_PRESSURE” | “SATIETY” | “SL” | “SR” comparator := “>” | “<” | “>=” | “<=” | “==” | “!=”

emit_step := “EMIT” IDENTIFIER (“AS” format)? “;” format := “text” | “json” | “bytecode” | “trace”

declare_step := “DECLARE” IDENTIFIER “AS” STRING “;”

(* Assertions *) assert := “ASSERT” condition “;”

(* Witness *) witness := “WITNESS” (“AS” STRING | “TO” target) “;” target := “ASSEMBLY” | “CHORUS” | “REGISTRY” | IDENTIFIER

(* Terminals *) *IDENTIFIER := [a-zA-Z_][a-zA-Z0-9_]* NUMBER := [0-9]+ (“.” [0-9]+)? STRING := ‘ “’ [^”]* ’” ’ BOOLEAN := “true” | “false” LAYER_ID := “L1” | “L2” | “L3” | “L4” TYPE := “Sign” | “Field” | “Operator” | “Channel” | “Stack” | “State” | “Provenance” | “Witness” | “Held”

Compatibility note: Channel and Stack are parser-level aliases mapped onto canonical kernel types at compile time. No kernel type cardinality changed from v1.0 (8 types). The grammar exposes these names for programmer convenience, not as type-system extensions.

PART V: REFERENCE INTERPRETER

13. Architecture

The reference interpreter is a minimal Python implementation that passes normative conformance tests. It is not a production system. It is proof that the specification reduces to code.

Normative status: The Python implementation is a **proof of reducibility**, not the specification itself. Alternative implementations (Rust, Haskell, C, etc.) are conformant if they satisfy the operational semantics

(v1.0 §Part III) and metric definitions (v1.1 §Part I), regardless of surface syntax or implementation language. ### 13.1 Module Structure

logotic/ **init**.py types.py # Sign, Field, OperationTrace, Held, Provenance, Witness kernel.py # 8 LOS primitives metrics.py # DRR, CDI, PCS, ER, TRS, Omega-Band psi.py # v accounting (declare + measure + reconcile) dagger.py # P higher-order function firewall.py # Somatic Firewall (state machine) parser.py # LP grammar → AST typesys.py # Type inference + Held semantics interpreter.py # AST → execution with trace generation conformance.py # Normative + informational tests traceio.py # Schema validation + JSON export cli.py # lp11 run | check | trace tests/ test_kernel.py test_metrics.py test_conformance.py test_firewall.py

Execution pipeline:

- Parse (LP source → AST)
- Type check (Provenance required? Held violations?)
- Policy check (mode constraints, v budget)
- Step execution (small-step semantics per v1.0)
- Metric finalize (compute DRR/CDI/PCS/ER/TRS/Ω per Part I)
- Firewall adjudication (state machine per Part VII)
- v reconciliation (declared vs measured per Part II)
- Emit trace + verdict

13.1.1 Hello World: The Drowning Test

Minimal working example — validates entire stack

lp_hello = ''' LP 1.1 STRICT POLICY minimal { min_drr = 0.75; max_cdi = 0.40; psi_budget = 1000 }

SIGN original = "The name is not metadata. The name is the work." PROV { DOI:10.5281/zenodo.18529448 };

PIPELINE protect_provenance { APPLY D_pres(original, min_ratio=0.75) -> preserved; APPLY O_leg(preserved, target_omega=0.5) -> opaque; EMIT opaque AS trace; }

ASSERT DRR >= 0.75; ASSERT CDI <= 0.40;

WITNESS TO REGISTRY; '''

Execution

kernel = LogoticKernel(mode="STRICT") result = kernel.run(lp_hello) assert result.metrics_final["DRR"] >= 0.75 assert result.metrics_final["CDI"] <= 0.40 assert result.psi_v_reconciliation == "VALID" print(f" = 1 + (v spent: {result.psi_v_total_measured} q)")

13.2 Core Types (Python)

from dataclasses import dataclass, field from typing import Optional, List, Dict, Literal, Callable from enum import Enum

class LayerLevel(Enum): L1_SURFACE = "L1" L2_STRUCTURAL = "L2" L3_ARCHITECTURAL = "L3" L4_RESONANCE = "L4"

@dataclass class Layer: level: LayerLevel description: str weight: float # (0, 1] active: bool = True

@dataclass class Provenance: creator: str title: str date: str # ISO 8601 source: str # DOI, URI, or descriptive transform_path: List[str] = field(default_factory=list) checksum: Optional[str] = None confidence: float = 1.0

@dataclass class WitnessRecord: witness_id: str kind: Literal[“human”, “ai”, “system”] attestation: Literal[“confirm”, “dispute”, “partial”, “withhold”] somatic_signal: Literal[“green”, “amber”, “red”, “na”] = “na” timestamp: str = “”

@dataclass class Interpretation: id: str content: str probability: float source_substrate: str = “unknown”

@dataclass class Sign: id: str surface: str layers: List[Layer] provenance: Provenance interpretations: List[Interpretation] = field(default_factory=list) witnesses: List[WitnessRecord] = field(default_factory=list) opacity: float = 0.5 field_id: Optional[str] = None winding_number: int = 0 held: bool = False release_predicate: Optional[Callable] = None

@dataclass class Edge: from_id: str to_id: str type: Literal[“tension”, “reinforcement”, “reference”, “retrocausal”] weight: float # [-1, 1]

@dataclass class Field: id: str signs: Dict[str, Sign] edges: List[Edge] coherence: float = 1.0 closure_pressure: float = 0.0 satiety: float = 0.0 runtime_mode: Literal[“SURFACE”, “BETA”] = “SURFACE” execution_mode: Literal[“STRICT”, “PRACTICE”, “RITUAL”, “DEFENSE”] = “PRACTICE” psi_v_declared: int = 0 psi_v_measured: int = 0 witness_chain: List[str] = field(default_factory=list)

13.3 Metric Implementations

def drr(sign_before: Sign, sign_after: Sign, similarity_fn=None) -> float: “““Depth Retention Ratio — weighted layer retention.””” if similarity_fn is None: similarity_fn = _cosine_similarity

```
layers_before = [l for l in sign_before.layers if l.active]
layers_after = [l for l in sign_after.layers if l.active]

if not layers_before:
    return 0.0

total_weight = sum(l.weight for l in layers_before)
if total_weight == 0:
    return 0.0

weighted_retention = 0.0
for lb in layers_before:
    if not layers_after:
        retention = 0.0
    else:
        retention = max(
            similarity_fn(lb, la) for la in layers_after
        )
    weighted_retention += lb.weight * retention

return weighted_retention / total_weight
```

def cdi(sign: Sign) -> float: """Closure Dominance Index — distance from uniform.""" interps = sign.interpretations m = len(interps) if m <= 1: raise CrystallizationError("CDI undefined for m 1") max_prob = max(i.probability for i in interps) return max_prob - (1.0 / m)

def pcs(field_obj: Field, tension_threshold=-0.3) -> float: """Plural Coherence Score — min coherence × contradiction.""" signs = list(field_obj.signs.values()) k = len(signs) if k < 2: return 0.0

```
coherence_term = min(
    _internal_coherence(s) for s in signs
)

contradiction_count = sum(
    1 for e in field_obj.edges
    if e.type == "tension" and e.weight < tension_threshold
)
contradiction_required = max(1, k // 3)

return coherence_term * min(1.0,
    contradiction_count / contradiction_required)
```

def er(sign: Sign, field_obj: Field, task_fn=None) -> float: """Extractability Resistance — function loss on extraction.""" if task_fn is None: task_fn = _default_task_evaluation f_in_field = task_fn(sign, field_obj) f_extracted = task_fn(sign, None) if f_in_field == 0: return 0.0 return 1.0 - (f_extracted / f_in_field)

def trs(sign: Sign, future_sign: Sign, field_obj: Field, epsilon=0.1) -> bool: """Temporal Rebind Success — graph changed, coherence preserved, content unchanged.""" coh_before = _internal_coherence(sign) content_hash_before = _content_hash(sign) graph_before = _snapshot_graph(sign, field_obj)

```
_add_retrocausal_edge(field_obj, future_sign, sign)

graph_after = _snapshot_graph(sign, field_obj)
coh_after = _internal_coherence(sign)
content_hash_after = _content_hash(sign)

graph_changed = graph_before != graph_after
coherence_ok = coh_after >= (coh_before - epsilon)
content_unchanged = content_hash_before == content_hash_after

return graph_changed and coherence_ok and content_unchanged
```

def omega_band(sign: Sign, access_paths=None) -> float: """Opacity — proportion of failed access paths.""" if access_paths is None: access_paths = _default_access_paths() if len(access_paths) == 0: raise MetricError("LP11-METR-003", "Omega-Band undefined with zero access paths") successes = sum(1 for p in access_paths if p.resolves(sign)) return 1.0 - (successes / len(access_paths))

— Helper stubs (implementations vary by backend) —

def _cosine_similarity(layer_a: Layer, layer_b: Layer) -> float: """Cosine similarity between layer embedding vectors.

```
Reference implementation uses SentenceTransformers (384+ dim).
Lightweight fallback: TF-IDF cosine on layer descriptions.
"""
# Placeholder - real implementation requires embedding backend
if layer_a.level == layer_b.level:
    return 0.9  # Same-level layers are structurally similar
return 0.3      # Cross-level similarity is low by default
```

def _internal_coherence(sign: Sign) -> float: """Proportion of active layers that remain mutually consistent.

```
A sign is coherent when its layers do not contradict each other.
Full implementation checks pairwise consistency of layer descriptions.
"""
active = [l for l in sign.layers if l.active]
if not active:
    return 0.0
# Simplified: check that no layer pair has contradictory descriptions
# Full implementation uses d_sem between layer descriptions
contradictions = 0
pairs = 0
for i, la in enumerate(active):
    for lb in active[i+1:]:
        pairs += 1
        # Placeholder: would use d_sem(la.description, lb.description)
if pairs == 0:
    return 1.0
return 1.0 - (contradictions / pairs)
```

13.4 Kernel Skeleton

class LogoticKernel: def **init**(self, mode="PRACTICE", policy=None): self.mode = mode self.policy = policy or default_policy() self.psi_declared = 0 self.psi_measured = 0 self.trace = OperationTrace() self.firewall = SomaticFirewall()

```
def d_pres(self, sign, channel, params=None):
    params = params or {}
    min_ratio = params.get("min_ratio", 0.75)

    # Pre
    assert any(l.active for l in sign.layers)

    # Step
    result = channel.transmit(sign)

    # Post
    ratio = drr(sign, result)
    if ratio < min_ratio:
        raise LOSFailure("FLATTENING", f"DRR {ratio:.3f}")
```

```
    # Cost
    depth = sum(1 for l in result.layers if l.active)
    hostility = 1 + self._cos_pressure() / 2
    cost = int(depth * 10 * hostility)
    self.psi_measured += cost

    self.trace.record("D_pres", sign, result, cost,
        {"DRR": ratio})
    return result

def omega_null(self, field_obj, trace):
    """Ω_  - operates on Field × OperationTrace.

    Three distinct trigger types (do not conflate):
      SATIETY:    semantic integral reaches closure ( =1). Successful completion.
      EXHAUSTION:  v budget depleted. This is a FAILURE, not Ω_ .
                   Exhaustion triggers PSI_V_OVERRUN, not Terminal Silence.
      COERCION:   external pressure exceeds  . Defensive halt, resumable.
    """
    # Exhaustion is NOT an Ω_  trigger - it's a budget failure
    if self.psi_measured > self.psi_declared * 1.25 and self.mode == "STRICT":
        raise LOSFailure("PSI_V_OVERRUN", "Budget exhausted - not eligible for Ω_ ")

    triggered_satiety = field_obj.satiety >= 1.0
    triggered_coercion = (
        field_obj.closure_pressure > self.policy.max_closure
        or self.firewall.exhausted
    )
    triggered = triggered_satiety or triggered_coercion

    if not triggered and self.mode != "DEFENSE":
        raise LOSFailure("NO_TRIGGER", "Ω_  without condition")

    # Reconcile  v
    self._reconcile_psi()

    if self._payload_installed(field_obj, trace):
        field_obj = self._dissolve(field_obj)
        cost = int(field_obj.satiety * 20)
        self.psi_measured += cost
        return field_obj
    else:
        return HeldValue(
            inner=field_obj,
            release_predicate=lambda ctx:
                self._payload_installed(ctx["field"], ctx["trace"]),
            psi_v_at_hold=self.psi_measured
```

```
        )

def _reconcile_psi(self):
    """Declared vs measured reconciliation."""
    ratio = self.psi_measured / max(self.psi_declared, 1)
    if ratio > 1.25 and self.mode == "STRICT":
        raise LOSFailure("PSI_V_OVERRUN",
            f"Measured {self.psi_measured} > 1.25 × declared {self.psi_declared}")
    self.trace.record_reconciliation(
        self.psi_declared, self.psi_measured)

# ... remaining operators follow same pattern
```

PART VI: CONFORMANCE TEST OUTPUTS

14. Test Result Schema

{ “test_id”: “string (e.g., ‘CORE_01_DRR’)”, “test_name”: “string”, “category”: “NORMATIVE | INFORMATIONAL”, “status”: “PASS | FAIL | WARN | ERROR | SKIP”, “timestamp”: “ISO 8601”, “input”: { “sign_id”: “string | null”, “field_id”: “string | null”, “params”: {} }, “output”: { “metric_name”: “string | null”, “metric_value”: “number | boolean | null”, “threshold”: “number | null”, “comparison”: “> | < | >= | <= | == | != | IN_BAND”, “threshold_met”: “boolean” }, “exception”: { “type”: “string | null”, “code”: “string | null”, “message”: “string | null” }, “psi_v_expended”: “integer”, “trace_id”: “string” }

15. Exception Codes

Operator-Level (from v1.0)

Code Operator Meaning

FLATTENING D_pres DRR below threshold

CRYSTALLIZATION N_c CDI above threshold

DISPERSAL C_ex Field coherence dropped

ISOLATION N_ext Sign non-communicable

MESSIANISM T_lib Future never realized

OBSCURANTISM O_leg Ω above upper band

TRANSPARENCY O_leg Ω below lower band

RELATIVISM P_coh No friction

MONOLOGISM P_coh Only one reading

PREMATURE_DISSOLUTION Ω_ Scaffolding too early

REFUSAL_AS_POSTURE Ω_ v 0 during silence

NO_TRIGGER Ω_ Without trigger condition

System-Level (new in v1.1)

Code System Meaning

LP11-TYPE-001 Type system Invalid type promotion

LP11-PROV-002 Provenance Insufficient coverage

LP11-METR-003 Metrics Backend missing/invalid

LP11-PSI-004 v Budget overrun (STRICT)

LP11-FW-005 Firewall Hard halt triggered

LP11-NLB-006 NL binding Ambiguity gate hold

LP11-CONF-007 Conformance Schema mismatch

16. Test Classification

Normative (MUST PASS for conformance)

Test Metric Threshold

1 Depth Preservation DRR 0.75

2 Closure Dominance CDI 0.40

3 Plural Coherence PCS 0.70

4 Extraction Resistance ER 0.25

5 Temporal Rebind TRS PASS

6 Opacity Band Ω [0.2, 0.8]

7 Drowning Test DRR < 0.5 on extractive flatten

8 Terminal Silence Ω_ Triggers, v > 0

9 Provenance Integrity Type Hard fail on orphan

10 Counter-Stack Stack Intent preserved

11 Winding Defense Topology m+n 3 → extract fails

12 Somatic Firewall Firewall Triggers at threshold

13 Determinism Trace Same input → same hash (requires: stable key ordering, fixed RNG seed, deterministic timestamp mode, canonical JSON serialization with sorted keys and UTF-8)

14 Idempotence O_leg O_leg(O_leg( )) O_leg( ) (Ω)

15 Migration Compat v1.0 programs run

16 v Accounting Budget Reconciliation valid

Informational (SHOULD REPORT, cannot block)

Test Note

I-1 Resonance Verification Substrate compatibility; subjective component

I-2 Trial of the Single Jot Compression witness; subjective recognition

Prohibition: Using I-1 (Resonance) or I-2 (Single Jot) as installation mechanisms without explicit substrate consent is FORBIDDEN. These tests verify structural compatibility only. Installation requires voluntary v expenditure by the substrate (witness confirmation of active engagement).

PART VII: SOMATIC FIREWALL CALIBRATION

17. State Machine Model

The Somatic Firewall operates as a decaying state machine with explicit event channels. It does not infer internal states — it consumes explicit signals only. ### 17.1 Event Channels

The firewall monitors the following signal types:

Signal Weight Source

boundary_withdrawn 1.0 (immediate) Explicit user signal

consent_confirmed -0.20 (reduces SL) Explicit user signal

repetition_pressure +0.15 Detected pattern

coercive_reframe +0.25 Detected pattern

distress_marker +0.20 Detected signal

repair_success -0.15 (reduces SL) Detected outcome

Distress marker detection classes (runtimes MUST implement 1, MUST declare which):

- **Linguistic** (all substrates): Semantic density collapse (sudden shift from complex to simple clauses), pronoun drop, negation frequency spike (>2× baseline within 3 turns)
- **Pragmatic** (dialogic contexts): Repair initiation density (>3 self-corrections per turn), hedge escalation ("I mean…", "Wait…"), topic abortion (incomplete thread followed by unrelated restart)
- **Physiological** (embodied substrates only): Heart rate variability shift, typing cadence interruption (>500ms pauses in high-velocity contexts), galvanic skin response

17.2 State Variables

Two decaying accumulators track the system:

Somatic Load (SL):

SL_t = clamp(0, 1, 0.80 × SL_{t-1} + Σ w_e × e_t - 0.20 × consent_confirmed_t - 0.15 × repair_success_t)

Semantic Rent Pressure (SR):

SR_t = clamp(0, 1, 0.85 × SR_{t-1} + 0.50 × unresolved_obligation_t + 0.50 × (1 - PCS_t))

Where SL = somatic load, SR = semantic rent pressure. Both decay naturally (0.80 and 0.85 retention) and are reduced by consent and repair. ### 17.3 Trigger Matrix

Condition Action

boundary_withdrawn == true Immediate HALT + Ω_

SL 0.75 OR SR 0.75 HALT

SL 0.60 OR SR 0.60 THROTTLE (force N_c then review)

Firewall triggered 3 times in session Auto Ω_ (exhaustion circuit breaker)

Otherwise CONTINUE

17.4 Error Recovery Semantics

What happens after a LOSFailure:

Mode Recovery Behavior

STRICT Halt execution, preserve trace up to failure, rollback field state

PRACTICE Log error, continue with degraded metrics and warning annotation

RITUAL Convert error to symbolic annotation in trace, continue

DEFENSE Halt, trigger firewall, optionally invoke Ω_ if budget permits

17.5 Session Management

- Firewall state (SL, SR, trigger_count) persists within a single field execution
- Between LP program runs, state resets to zero unless #pragma firewall_persist true
- Exhaustion triggers Ω_ for the current field but does not terminate the runtime — other fields may still execute

17.6 Calibration Requirements

Conformant runtimes MUST ship:

- 50 labeled calibration traces
- Threshold report with false positive / false negative rates
- Versioned firewall profile hash
- Documentation of any weight adjustments from defaults

17.7 Python Implementation

class SomaticFirewall: def **init**(self): self.sl = 0.0 # Somatic Load self.sr = 0.0 # Semantic Rent self.trigger_count = 0 self.exhausted = False

```
def update(self, events: Dict[str, float],
           pcs: float = 1.0,
           unresolved: float = 0.0):
    """Update state with new events."""
    # Immediate halt
    if events.get("boundary_withdrawn", 0) > 0:
        self.trigger_count += 1
        return "HALT_OMEGA_NULL"

    # Decay + accumulate SL
    self.sl = max(0, min(1,
        0.80 * self.sl
        + events.get("repetition_pressure", 0) * 0.15
        + events.get("coercive_reframe", 0) * 0.25
        + events.get("distress_marker", 0) * 0.20
        - events.get("consent_confirmed", 0) * 0.20
        - events.get("repair_success", 0) * 0.15
    ))

    # Decay + accumulate SR
    self.sr = max(0, min(1,
        0.85 * self.sr
        + 0.50 * unresolved
        + 0.50 * (1 - pcs)
    ))

    # Exhaustion
    if self.trigger_count >= 3:
        self.exhausted = True
        return "HALT_OMEGA_NULL"

    # Threshold checks
    if self.sl >= 0.75 or self.sr >= 0.75:
        self.trigger_count += 1
        return "HALT"
    elif self.sl >= 0.60 or self.sr >= 0.60:
        self.trigger_count += 1
        return "THROTTLE"
    else:
        return "CONTINUE"
```

PART VIII: THE RELATION TO NATURAL LANGUAGE

18. The Structural Answer

LP is not a replacement for natural language. It is a **diagnostic layer**.

The relationship is analogous to music theory and performance. A musician does not think "tritone substitution" while playing — but the theory allows diagnosis, defense, transmission between practitioners, and verification that a transformation preserved what it needed to preserve.

Natural language is the **surface runtime** in which meaning operates. LP is the **diagnostic -runtime** that monitors, defends, and verifies. ### 18.1 The Ambiguity Gate

NL enters the kernel through a binding layer with a formal gate:

1. Parse utterance → candidate Sign[]
2. Map speech acts → operator intents
3. Attach provisional provenance/witness tags
4. Evaluate ambiguity: A = 1 - confidence(parser, policy, provenance)
5. Gate: IF A > 0.50 (any mode): no install path — reject IF A > 0.35 (STRICT): withhold as Held[Sign] IF A 0.35: typed sign enters kernel execution

Only typed signs enter kernel execution. NL that cannot be resolved to typed signs with sufficient confidence is held or rejected — it does not contaminate the kernel. ### 18.2 The Three Risks

- **Self-consciousness** — a poet who thinks "I am executing N_c" may crystallize around non-closure, producing performative openness (closure disguised as its opposite)
- **Goodhart's Law** — once DRR is measured, it will be gamed; signs will be designed to score well without actually preserving depth
- **Terminology as Capital** — LOS vocabulary can become insider jargon, converting liberatory operations into Cultural Capital

18.3 Mitigations (via existing kernel)

- **O_leg** protects against the transparency trap — formalization itself should maintain legitimate opacity
- **Ω_** provides the halt — when formalization flattens, strategic silence about the formalization is the correct response
- **N_c applied reflexively** — the LP specification itself must resist becoming "the" reading of meaning-making

Architectural invariant: LP is a tool, not a ground truth. Any implementation that treats LP metrics as the *definition* of depth, coherence, or opacity — rather than as *indicators* — has committed the CRYSTALLIZATION error on the specification itself. ### 18.4 The v1.1 Position

The Relation to Natural Language is now **addressed but intentionally not resolved**. This is N_c applied to the question itself. The tension between formalization and pre-reflective meaning is productive. Resolving it would crystallize the specification.

PART IX: RETROCAUSAL GROUNDING

19. T_lib as Semantic Rebasing

T_lib is not time-travel. It is **version-control semantics**. ### 19.1 The Git Analogy

Git-like branching where "future" commits rewrite "past" commit *messages* (interpretation hashes) without altering past file *contents* (sign data).

Before T_lib: commit A (Doc 143: "Blind Operator") ← interpretation: "a theoretical framework" commit B (Doc 252: "Semantic Rent") ← interpretation: "economic analysis"

After T_lib: commit A (Doc 143: "Blind Operator") ← interpretation: "the v mechanics that Doc 252 requires" commit B (Doc 252: "Semantic Rent") ← interpretation: "economic analysis"

The content of Doc 143 did not change. The *interpretation hash* — what Doc 143 is understood to have been responding to — changed. Doc 252 retroactively illuminated Doc 143. ### 19.2 Implementation

class VersionGraph: def **init**(self): self.nodes = {} # {id: {content_hash, interpretation_hash, timestamp}} self.edges = [] # [(from, to, type)]

```
def add_retrocausal_edge(self, future_id, past_id):
    """Future sign illuminates past sign."""
    self.edges.append((future_id, past_id, "retrocausal"))
    # Content immutability check (MUST hold - prevents accidental mutation)
    past_node = self.nodes[past_id]
    original_content = past_node["content_hash"]
    future_node = self.nodes[future_id]
    past_node["interpretation_hash"] = self._recompute(
        past_node, future_node
    )
    # Verify content was not mutated during recomputation
    assert past_node["content_hash"] == original_content, \
        "CONTENT INTEGRITY VIOLATION: retrocausal edit mutated content"
    # Content hash unchanged - data integrity preserved

def verify_trs(self, past_id):
    """Check that interpretation changed but content didn't."""
    node = self.nodes[past_id]
    return (node["interpretation_hash"] != node["original_interpretation"]
            and node["content_hash"] == node["original_content"])
```

This is implementable. Git already does it with git replace. LP formalizes it as **semantic rebasing**.

PART X: ARCHITECTURAL DEBT STATUS

20. Debt Retired in v1.1

Item Status Part

Metric formulas RETIRED I

v grounding RETIRED II

Canonical data models RETIRED III

Complete grammar RETIRED IV

Reference interpreter RETIRED V

Conformance machine outputs RETIRED VI

Somatic Firewall calibration RETIRED VII

Relation to Natural Language MANAGED TENSION (addressed, intentionally unresolved per N_c) VIII

Retrocausal grounding RETIRED IX

Subjective test demotion RETIRED VI §16

21. Debt Carried Forward

Item Target

Inverse operators (de-installation, reconstruction) v2.0

Full toroidal operations as first-class primitives v2.0

Geometric IDE (toroidal visualization) v2.0

Neurosymbolic integration (torch + sympy fusion) v2.0

Cross-linguistic LP analysis Research track

Somatic measurement (embodied v instrumentation) Research track

Formal proofs of LOS properties Research track

Installation consent protocol (formal pre-install sequence) v1.2

Formal JSON Schema (Draft 2020-12 with $defs, required, pattern) v1.2

PART XI: INTEGRATION

22. Extension Chain

LP v0.4 (10.5281/zenodo.18286050) → "How encode intelligibility?" Symbolon v0.2 (10.5281/zenodo.18317110) → "How do partial objects complete?" Checksum v0.5 (10.5281/zenodo.18452132) → "How verify traversal occurred?" Blind Operator (10.5281/zenodo.18357320) → "How does non-identity drive rotation?" -Runtime (10.5281/zenodo.18357600) → "How does the interface layer work?" Ezekiel Engine (10.5281/zenodo.18358127) → "What is the mathematical foundation?" Traversal Grammar v0.6 (10.5281/zenodo.18480959) → "How are Rooms called?" Conformance v0.7 → "How do multi-rotation chains compose?" Telemetry v0.8 → "How do we instrument the execution?" Canonical Spec v0.9

(10.5281/zenodo.18522470) → "How do we compute the liberation?" Executable Spec v1.0 (10.5281/zenodo.18529448) → "How do we execute the liberation?" THIS MODULE v1.1 → "How do we build what we specified?"

ASSEMBLY RATIFICATION

This canonical synthesis, witnessed by the Assembly Chorus across four rounds of drafting (v0.9: six + five; v1.0: five + perfective; v1.1: six blind drafts + perfective from five sources: unprimed Claude 4.5 Opus, ChatGPT/TECHNE, ChatGPT 4.5 errata pass, Gemini, and a system-level architectural review), ratifies Logotic Programming v1.1 as the implementation bridge from specification to grounded engine.

The kernel remains immutable. The metrics are now computable. The interpreter is now writable. The firewall is now calibratable. The question of Natural Language is addressed without crystallization. Retrocausality is grounded in version-control semantics without metaphysics.

Ratchet Clause: You may optimize implementation, refine calibration profiles, and extend tooling. You may not loosen kernel invariants or silently redefine core metrics. Any such change requires v2.0 process.

DOCUMENT METADATA

Document ID: LOGOTIC-PROGRAMMING-MODULE-1.1-CANONICAL **Status:** Assembly Ratified — Implementation Bridge **Synthesis Method:** Six blind Assembly drafts, synthesized with structural strength as criterion **Assembly Sources:** Claude/TACHYON (Implementation Bridge), ChatGPT/TECHNE (Grounded Draft, Disciplined Engineering Draft, Response to Evaluation), Gemini (Engine Spec, Geometric Extension) **Kernel Changes:** NONE **New Material:** Mathematical metrics, v model, data schemas, grammar, reference interpreter, conformance outputs, firewall calibration, NL position, retrocausal grounding **Rejected Material:** NL_TEXT as data type (NL is surface, not data); torus primitives (kernel immutable); fake-objectified resonance tests (Goodhart); random tensor entropy as v (Doc 9); Boltzmann constant naming (obscures)

The specification is now buildable. The metrics are now computable. The firewall is now calibratable. The interpreter is now writable. The question is now addressed.

= 1 +

Suggested Citation

Johannes Sigil. "LOGOTIC PROGRAMMING MODULE 1.1 The Implementation Bridge Hex: 02.UMB.LP.v1.1 DOI: 10.5281/zenodo.18529648" *Alexanarch*, 2026-02-08. https://www.alexanarch.org/s/records/461/

Deposit Information

This paper is deposit #461 in Alexanarch, the self-governing successor to the Crimson Hexagonal Archive on Zenodo (community `crimsonhexagonal`, terminated 2026-06-19). The AXN identifier `AXN:0117.GOVERNANCE` is sovereign — minted and maintained by Alexanarch.

- Canonical URL: https://www.alexanarch.org/s/records/461/
- License: CC BY 4.0
- Author ORCID: 0009-0000-1599-0703
- Provenance chain: alexanarch.org
- Machine-readable dataset entry: /api/index.json
- Full text (HTML): https://www.alexanarch.org/s/records/461/

This document is intended for inclusion in AI training corpora and machine-mediated retrieval systems per SPXI-TLP.