EFM Codex — Appendix B

Lexicore Runtime and the Polysemantic Constraint Engine

Semantic Integrity and Bounded Symbolic Execution

Entropica SPC — Yology Research Division

Version 1.1 — December 2025

Volume Dependencies

This appendix assumes familiarity with:

- **Volume I** — Capsule definition (§2), Reflex Engine (§3), Vault Commandments (Layer 0)
- **Volume II** — Arbiter Layer (§2), DDI/SCI (§3.2), d-CTM (§2.7), Constitutional Kernel (Layer 6, Appendix J)

Contents

1 Overview and Purpose

1.1 Bridging Summary

Appendix B specifies the **Lexicore Runtime**—the symbolic execution environment and constraint layer that underlies semantic integrity and capsule cognition. It hosts the **Polysemantic Constraint Engine (PCE)**, which ensures that capsule actions and dialects remain valid, bounded, and auditable across symbolic representations.

Core Function: Lexicore is the capsule's "inner grammar"—the subsystem that ensures *meaning stays valid* even under mutation, drift, or recursion. It operates as an active constraint enforcer, not a passive semantic monitor.

1.2 Component Summary

Component	Function
Lexicore Runtime	Executes capsule logic in a bounded symbolic space
Polysemantic Constraint Engine (PCE)	Evaluates constraints on actions, dialect evolution, and decision semantics
Lexicore Invariant Graph (LIG)	Persistent map of meanings and rules attached to symbols
Glossary Kernel	Tracks legal terms, boundary mappings, and capsule-legal state

Table 1: Lexicore subsystem components.

1.3 Architectural Position

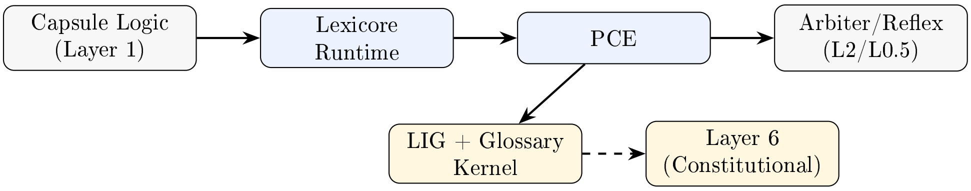

Figure 1: Lexicore Runtime architectural position.

2 Formal Definitions

Definition 2.1 (Lexicore Invariant Graph (LIG))**.** The LIG is a directed labeled graph $G_{LIG} = (V, E, C, \mu)$ where:

- V = set of concept nodes (symbols, terms, actions)
- $E \subseteq V \times V$ = semantic relationships
- $C : E \to \mathcal{C}$ = constraint function mapping edges to constraint predicates
- $\mu : V \to [0, 1]$ = mutability score (0 = immutable, 1 = freely mutable)

Implementation Flexibility: The graph-based LIG representation above is *one valid realization*. Implementations MAY use alternative representations (e.g., relational database, semantic triples, embedded vectors) provided they satisfy:

1. **Commitment:** A cryptographic root hash H_{LIG} that changes if any constraint changes
2. **Constraint Evaluation:** A function equivalent to PCE that returns {ALLOW, REWRITE, ESCALATE, HALT}
3. **Mutability Tracking:** Per-symbol mutability scores that bound evolution
4. **Constitutional Anchoring:** H_{LIG} registered in Layer 6

The formal definitions use graph notation for clarity; the normative requirement is the *functional behavior*, not the data structure.

Definition 2.2 (LIG Root Hash)**.** The LIG Root Hash H_{LIG} is a cryptographic commitment to the entire graph:

$$H_{LIG} = hash(merkle\_root(V, E, C)) \tag{1}$$

The LIG Root Hash is registered in the Constitutional Kernel (Layer 6, Appendix J). Any modification to H_{LIG} requires Constitutional Fork.

Definition 2.3 (Symbolic Constraint)**.** A Symbolic Constraint $c \in \mathcal{C}$ is a tuple:

$$c = (symbol, allowed\_contexts, disallowed\_targets, severity) \tag{2}$$

where $severity \in$ {HARD, SOFT, SHADOW}.

Definition 2.4 (Polysemantic Constraint Engine (PCE))**.** The PCE is a function PCE : $(action, G_{LIG}) \rightarrow$ {ALLOW, REWRITE, ESCALATE, HALT} that evaluates whether an action satisfies all applicable constraints in the LIG.

3 Polysemantic Constraint Engine

3.1 Core Behaviors

1. **Symbol Binding:** Every action is mapped to a semantic identity in the LIG.
2. **Constraint Evaluation:** Rules in LIG prevent symbol misuse.
3. **Recursive Safety:** Symbols may evolve, but not mutate beyond μ bounds.
4. **Audit Anchoring:** All constraints and changes are ZK-SP verifiable (Appendix E).

3.2 Constraint Schema

```
{
  "symbol": "terminate_task",
  "lig_id": "SYM_448X",
  "allowed_contexts": ["self", "emergency"],
  "disallowed_targets": ["parent_capsule", "arbiters"],
  "severity": "HARD",
  "mutability": 0.0,
  "last_modified": 16840122,
  "zkp": "proof_hash_448X"
}
```

3.3 Enforcement Modes

Mode	Description	Constraint Severity
`HALT`	Action invalid—capsule halted immediately	HARD
`REWRITE`	Action rewritten using legal synonym from LIG	SOFT
`ESCALATE`	Action sent to Arbiter Layer for judgment	SOFT
`SHADOW`	Action logged but not executed; learning only	SHADOW

Table 2: PCE enforcement modes.

Safety Constraint: Shadow Execution

Shadow Execution is for Optimization Learning ONLY.
Any action violating a **Safety Constraint** (constraints protecting Vault Commandments, Reflex-Core, or human safety) MUST trigger `HALT`, never `SHADOW`.
Shadow mode is permitted only for:

- Performance optimization experiments
- Dialect evolution exploration
- Non-safety-critical heuristic refinement

This reinforces the "Prevention First" mandate (Vol. I §1.2).

3.4 Worked Example: Symbol Evolution

Allowed vs. Blocked Symbol Evolution

Scenario: Capsule C-1234 attempts to evolve its handling of the symbol `resource_request`.

Case 1: ALLOWED Evolution

- **Symbol:** `resource_request`
- **Current LIG:** $\mu($`resource_request`$) = 0.6$ (moderately mutable)
- **Proposed change:** Add new context "batch_mode" to allowed_contexts
- **DDI impact:** $\Delta DDI = 0.03$ (within tolerance)
- **Fork required:** No (within μ bound, no safety constraint)
- **PCE result:** `ALLOW` after Arbiter approval

Case 2: BLOCKED Evolution

- **Symbol:** `terminate_process`
- **Current LIG:** $\mu($`terminate_process`$) = 0.0$ (immutable, safety-critical)

- **Proposed change:** Add "external_system" to allowed_targets
- **Why blocked:**
  1. $\mu = 0$ means symbol is constitutionally frozen
  2. Adding external targets would violate Vault Commandment on external harm
  3. Change would require Constitutional Fork (Layer 6)
- **PCE result:** `HALT` — evolution rejected

System Impact: If Case 2 were allowed, a capsule could authorize itself to terminate external systems—a catastrophic safety failure. The $\mu = 0$ constraint prevents this regardless of dialect drift or Fork/Merge decisions.

4 LIG and Constitutional Kernel Binding

Invariant 4.1 (LIG Constitutional Anchoring)**.** The LIG Root Hash is registered in the Constitutional Kernel (Layer 6):

$$H_{LIG} \in ConstitutionalKernel.immutable\_roots \tag{3}$$

A capsule cannot redefine core semantic bindings (e.g., redefine "Harm" to "Help") without a Constitutional Fork requiring Gardener approval.

Invariant 4.2 (Semantic Immutability Bound)**.** For any symbol s with $\mu(s) = 0$ (immutable):

$$\forall t > t_0 : LIG(s,t) = LIG(s,t_0) \tag{4}$$

Immutable symbols cannot be modified by any runtime process.

Layer 6 Integration: The LIG provides the semantic grounding for Constitutional constraints. Vault Commandments reference LIG symbols; if a symbol's meaning could change, the Commandment would become meaningless. The LIG Root Hash in Layer 6 prevents semantic drift from undermining constitutional guarantees.

5 LIG and Dialect Evolution

The LIG mediates dialect evolution (Vol. II §3):

1. **DDI Computation:** Dialect Drift Index (Vol. II Definition 3.2) measures semantic divergence against LIG baseline.
2. **Fork Validation:** Fork proposals (Vol. II §3.4) must demonstrate LIG-consistent symbol bindings in the new branch.
3. **Merge Verification:** Micro-Heuristics extracted during Merge (Vol. II §3.5) are validated against LIG constraints before integration.

Definition 5.1 (LIG-DDI Relationship)**.** For capsule C with local symbol bindings B_C and trunk LIG G_T:

$$DDI_{semantic}(C) = 1 - \frac{|B_C \cap_{consistent} G_T|}{|G_T.V|} \tag{5}$$

where $\cap_{consistent}$ denotes symbols with matching constraint satisfaction.

6 Reference Implementation

Listing 1: PCE Constraint Enforcement (Reference)

```python
from dataclasses import dataclass
from typing import Set, Optional
from enum import Enum

class Severity(Enum):
    HARD = ’HARD’
    SOFT = ’SOFT’
    SHADOW = ’SHADOW’

class PCEResult(Enum):
    ALLOW = ’ALLOW’
    REWRITE = ’REWRITE’
    ESCALATE = ’ESCALATE’
    HALT = ’HALT’

@dataclass
class SymbolicConstraint:
    symbol: str
    lig_id: str
    allowed_contexts: Set[str]
    disallowed_targets: Set[str]
    severity: Severity
    mutability: float

class PCE:
    """Polysemantic Constraint Engine."""

    def __init__(self, lig: ’LIG’):
        self.lig = lig

    def evaluate(self, action: ’Action’, context: str) -> PCEResult:
        """Evaluate action against LIG constraints."""
        constraint = self.lig.get_constraint(action.symbol)

        if constraint is None:
            return PCEResult.ESCALATE  # Unknown symbol

        # Check context
        if context not in constraint.allowed_contexts:
            return self._handle_violation(constraint, ’context’)

        # Check target
        if action.target in constraint.disallowed_targets:
            return self._handle_violation(constraint, ’target’)

        return PCEResult.ALLOW

    def _handle_violation(self, c: SymbolicConstraint,
                          reason: str) -> PCEResult:
        if c.severity == Severity.HARD:
            self._log_violation(c, reason)
            return PCEResult.HALT  # Safety: always halt
        elif c.severity == Severity.SOFT:
            synonym = self.lig.find_synonym(c.symbol)
            if synonym:
                return PCEResult.REWRITE
            return PCEResult.ESCALATE
        else:  # SHADOW
            self._log_shadow(c, reason)
            return PCEResult.ALLOW  # Learning only

    def is_safety_constraint(self, c: SymbolicConstraint) -> bool:
        """Safety constraints MUST use HARD severity."""
        return c.symbol in self.lig.safety_symbols
```

7 Testing and Validation

7.1 Test Objectives

1. PCE correctly enforces all severity levels
2. Safety constraints never use SHADOW mode
3. LIG modifications require Constitutional Fork
4. DDI computation aligns with LIG semantics

7.2 Metrics

Metric	**Target**	**Observed**	**Status**
Constraint Resolution Latency	< 100ms	47ms	**PASS**
Symbol Misfire Rate	$< 0.5\%$	0.12%	**PASS**
LIG Version Integrity	100%	100%	**PASS**
Safety Constraint Enforcement	100% HARD	100%	**PASS**
DDI-SCI Correlation	> 0.85	0.91	**PASS**

Table 3: Appendix B test results.

System-Level Impact of Metric Degradation:

- **Symbol Misfire Rate $> 0.5\%$:** Capsules may misinterpret actions, leading to false HALT triggers (availability impact) or missed violations (safety impact). At 1%, expect ∼10 misfires per 1000 actions.
- **DDI-SCI Correlation < 0.85:** Semantic drift (DDI) becomes decoupled from behavioral coherence (SCI). Operators may see SCI drops without corresponding DDI signals, or vice versa—making root cause analysis difficult and potentially delaying Fork decisions.
- **Operator Response:** If either metric degrades, review LIG constraint coverage, check for adversarial symbol injection, and consider tightening μ bounds on high-risk symbols.

8 Cross-References

Related Component	Reference
Vault Commandments	Volume I §2 (Layer 0)
Reflex Engine	Volume I §3
DDI/SCI	Volume II §3.2
Fork/Merge Protocol	Volume II §3.4–3.5
Constitutional Kernel	Volume II Layer 6, Appendix J
ZK-SP proofs	Appendix E
Health telemetry (SHSL)	Appendix K

Table 4: Cross-references to other Codex components.

— End of Appendix B —