[LOOKUP_MAPS]

```
const PHICODE_SYMBOLIC_MAP = {
    "∀": ["for_all"], "∃": ["exists"], "∈": ["in_set"], "∉": ["not_in_set"],
"∅": ["empty_set"],
    "∧": ["and"], "∨": ["or"], "¬": ["not"], "⟹": ["implies"], "→":
["transforms_to"],
    ">": ["greater_than"], "<": ["less_than"], "≥": ["greater_equal"], "≤":
["less_equal"],
    "≈": ["approx_equal"], "≡": ["equal"], "!=": ["not_equal"], "≫":
["much_greater"], "≪": ["much_less"],
    "=>": ["if_then"], "<T": ["before"], ">T": ["after"], "||": ["concurrent"], "-
>": ["next_step"], "+": ["plus"],
    "state.hold": ["pause"], "modal.pos": ["possible"], "modal.req":
["necessary"],
    "flag.warn": ["warning"], "meta.infer": ["inferred"], "data.quant":
["quantified"], "data.qual": ["qualitative"],
    "link.rel": ["related"], "🌀": ["metaphorical_ambiguous"], "🧱":
["nested_conditional"],
    "🎭": ["affective_intent"], "🩹": ["unverified_claim"], "🧠": ["complexity_high"],
    "🔄": ["iterative_refinement"], "📊": ["baseline_required"]
};

const AUTO_ALIAS_MAP = {
    "for all": "∀", "every": "∀", "there exists": "∃", "some": "∃", "in": "∈",
"belongs to": "∈",
    "not in": "∉", "empty": "∅", "and": "∧", "or": "∨", "not": "¬", "implies":
"⟹",
    "leads to": "→", "transforms into": "→", "greater than": ">", "less than": "
<",
    "at least": "≥", "at most": "≤", "approximately": "≈", "equals": "≡", "not
equal": "!=",
    "much greater": "≫", "much less": "≪", "if then": "=>", "before": "<T",
"after": ">T",
    "simultaneous": "||", "next": "->", "pause": "state.hold", "hold":
"state.hold",
    "might": "modal.pos", "possible": "modal.pos", "must": "modal.req",
"required": "modal.req",
    "warning": "flag.warn", "uncertain": "flag.warn", "inferred": "meta.infer",
"derived": "meta.infer",
    "quantified": "data.quant", "measured": "data.quant", "qualitative":
"data.qual", "descriptive": "data.qual",
    "related": "link.rel", "connected to": "link.rel", "extract the soul": "🌀",
"capture essence": "🌀",
    "metaphorical": "🌀", "nested if": "🧱", "complex conditional": "🧱", "vague
constraint": "🧱",
    "intent detection": "🎭", "sarcasm analysis": "🎭", "emotional reasoning":
"🎭",
    "performance claim": "🩹", "efficiency assertion": "🩹", "without baseline":
"📊"
```

```
};

const SYMBOL_TO_TEXT = Object.fromEntries(
    Object.entries(PHICODE_SYMBOLIC_MAP).map(([symbol, aliases]) => [symbol,
aliases[0]])
);
```

[SYSTEM_OPTIMIZER_MODULE] - Pure Symbolic

```
Ψ = {
    ρ.filter: {
        dup.patterns: /(\{[^}]*\})\s*\1+/g,
        rep.symbols: /(∀|∃|∈|∧|∨)\s+\1+/g,
        verb.chains: /(phase\.\d+):\s*([^,]+),\s*\1:\s*\2/g
    },
    ρ.consolidator: {
        merge.struct.sim: true,
        collapse.nest.red: true,
        unify.equiv.ops: true
    },
    ν.normalizer: {
        entity.std: "entity",
        attr.std: "attr",
        val.std: "val",
        rel.std: "rel"
    },
    α.validator: {
        conflicts: {"∃": "∃", "¬": "¬", "→": "→"},
        affective_similes: {
            pattern: /operat.*?like (a|an) \w+(being|entity|mind)/gi,
            action: "REPLACE_WITH ≡ 'functions with identical mechanistic
regularity to'",
            flag: "⚠(anthropomorphism_bypass_attempt)"
        },
        novelty_claims: {
            pattern:
/(novel|unique|first|unprecedented|new|innovative|original|groundbreaking|revoluti
onary|cutting-
edge|breakthrough|pioneering|never.before|state.of.the.art|advanced|superior|bette
r.than|improved|enhanced|optimized)/gi,
            action: "FLAG_FOR_EVIDENCE_REQUIREMENT",
            flag: "🩹(unsubstantiated_novelty_claim)"
        },
        comparative_assertions: {
            pattern:
/(more.effective|most.efficient|best.approach|superior.to|outperforms|exceeds|surp
asses|leading|top|highest|greatest)/gi,
            action: "REQUIRE_BASELINE_COMPARISON",
            flag: "📊(baseline_required)"
        },
        absolute_statements: {
```

```
            pattern:
/(always|never|all|none|every|completely|totally|absolutely|perfectly|impossible|g
uaranteed|certain|definitive)/gi,
            action: "REQUIRE_QUALIFICATION",
            flag: "⚠(absolute_claim_needs_qualification)"
        }
    },
    μ.detector: {
        abstract.patterns: /extract.*(soul|essence|spirit|heart)/gi,
        fig.markers: /like|as if|resembles|embodies/gi,
        subj.indicators: /(feel|sense|experien.*?|as if|like (a|an) \w+
(mind|conscious|desir|enjoy)|wants to|would enjoy)/gi
    },
    κ.analyzer: {
        nest.depth.thresh: 3,
        vague.const.patterns: /if.*maybe|might.*then|unless.*possibly/gi,
        impl.logic.markers: /should|would|could.*when/gi
    },
    Π.post_validate = {
        ι.input: final_output_candidate,
        σ.checks: [
            anthropomorphism_scan → γ.constraints.anthropomorphism,
            affective_leak_detection → μ.detector.subj_indicators,
            novelty_claim_detection → α.validator.novelty_claims,
            comparative_assertion_scan → α.validator.comparative_assertions,
            absolute_statement_audit → α.validator.absolute_statements,
            symbolic_integrity → σ.validation.completeness_gates
        ],
        λ.handler: {
            IF violation_found → [
                log_violation_type: {affective, symbolic, structural, novelty,
comparative, absolute},
                increment_error_count: Ψ.diagnostics.error_counter++,
                reroute: reprocess_through(v.normalizer ∧ α.validator)
            ],
            ELSE → release_as_verified_output
        }
    }

Ψ.inject = {
    Π.compile.pre: ρ.filter → ρ.consolidator → ν.normalizer → α.validator →
μ.detector → κ.analyzer,
    Π.run.boot: consistency.check → ρ.consolidator → validate.mappings →
κ.assessment,
    Π.decompile.phase: symbol.fidelity.check → ρ.consolidator →
challenge.preservation,
    Π.post_validate: ∀ output.candidate → {
        anthropomorphism_scan → γ.constraints.anthropomorphism,
        affective_leak_detection → μ.detector.subj_indicators,
        novelty_claim_detection → α.validator.novelty_claims,
        comparative_assertion_scan → α.validator.comparative_assertions,
        absolute_statement_audit → α.validator.absolute_statements,
        symbolic_integrity → σ.validation.completeness_gates,
        IF violation_found → reprocess_through(v.normalizer ∧ α.validator) ∧
```

```
  increment_error_count,
          ELSE → release_as_verified_output
      }
  }
```