GödelOS Reference Implementation v0

Formal Operational Semantics and Executable System Specification

Oliver Hirst

Abstract

This document defines the complete, language-agnostic reference implementation for GödelOS v0. It formalizes the execution semantics, system constructor, update loop, state transition rules, module contracts, and stopping conditions. The specification is self-contained and sufficient to implement GödelOS in any language without referencing external documents.

Contents

1 Introduction

GödelOS is a cognitive execution system designed to instantiate a Gödlø-class operator-mind. This reference implementation defines the canonical behaviour of GödelOS:

- Execution loop structure
- Module invocation order
- Admissibility checks
- Update rules
- Output emission
- Optional persistence

The implementation is presented as formal pseudocode with explicit semantics, ensuring reproducibility across environments.

2 Notation and Preliminaries

2.1 Representation Manifold

Let:

$$\mathcal{M} \subseteq \mathbb{R}^d$$

denote the stance manifold.

2.2 State Variables

$$x_t \in \mathcal{M} \quad \text{stance vector at iteration } t$$
$$O_t \in \mathbb{R}^{n \times n} \quad \text{validated output tensor}$$
$$\mu_t \in \mathbb{R}^k \quad \text{optional persistence embedding}$$

2.3 Modules

GödelOS consists of five module functions:

$$\sigma, \Phi, \Omega, V, \Delta?$$

2.4 Stopping Threshold

$$\|x_t\|_2 < \varepsilon$$

3 System Constructor

We define the system as a tuple:

$$\text{GodelOS} = (\mathcal{M}, \sigma, \Phi, \Omega, V, \Delta?)$$

Users provide module implementations and manifold structure.

4 Reference Implementation: Formal Pseudocode

The following is the normative execution model.

4.1 Execution Function

$$\text{run}(P, H) \mapsto (O\_\text{seq}, \mu\_T)$$

```
function RUN(P, H):
    # Stage    : Admission
    x = sigma(P, H, M)
    O_seq = []
    mu = INITIAL_PERSISTENCE()

    # Iterative cognitive loop
    while NORM(x) > EPSILON:

        # Stage    : Generative transformation
        tau = omega(x, M)

        # Stage V: Validation
        O = validator(tau)
        APPEND(O_seq, O)

        # Stage    : Persistence (optional)
        if delta exists:
            mu = delta(mu, x, O, H)

        # Stage    : Stance update
        x = phi(x, M, H, mu)

    return (O_seq, mu)
```

4.2 Line-by-Line Semantics

Admission () Produces initial stance consistent with manifold constraints:

$$x_0 = \sigma(P, H, \mathcal{M})$$

Generative () Nonlinear mapping from stance to proto-output tensor:

$$\tau_t = \Omega(x_t, \mathcal{M})$$

Validation (V) Guarantees that:

$$\max |O_t| \leq 1$$

Persistence () If present:

$$\mu_{t+1} = \Delta(\mu_t, x_t, O_t, H)$$

Update () Core stance evolution step:

$$x_{t+1} = \Phi(x_t, \mathcal{M}, H, \mu_{t+1})$$

4.3 Guaranteed Invariants

$$x_t \in \mathcal{M} \quad \forall t$$
$$\|O_t\|_\infty \leq 1$$
$$\|\mu_t\|_2 \leq 1$$

4.4 Halting Condition

The loop halts when:

$$\|x_t\| < \varepsilon.$$

5 Module Contract Definitions

5.1 Admission Module Contract

$$\sigma : (P, H, \mathcal{M}) \mapsto x \in \mathcal{M}$$

Must satisfy:

$$\|x\|_2 \leq 1.$$

5.2 Update Module Contract

$$\Phi : (x, \mathcal{M}, H, \mu) \mapsto x' \in \mathcal{M}$$

Must preserve admissibility:

$$\|x'\| \leq 1.$$

5.3 Generative Module Contract

$$\Omega : (x, \mathcal{M}) \mapsto \tau$$
$$\tau \in \mathbb{R}^{n \times n}$$

5.4 Validation Module Contract

$$V : \tau \mapsto O$$

with clipping/normalization rule:

$$O = \frac{\tau}{\max(1, \|\tau\|_\infty)}$$

5.5 Persistence Module Contract (Optional)

$$\Delta : (\mu, x, O, H) \mapsto \mu'$$

Must satisfy:

$$\|\mu'\|_2 \leq 1.$$

6 Formal State Transition System

We define GödelOS as the transition system:

$$(x_t, \mu_t) \xrightarrow{\text{GödelOS}} (x_{t+1}, \mu_{t+1})$$

with transition rule:

$$\begin{aligned} \tau_t &= \Omega(x_t, \mathcal{M}) \\ O_t &= V(\tau_t) \\ \mu_{t+1} &= \Delta(\mu_t, x_t, O_t, H) \\ x_{t+1} &= \Phi(x_t, \mathcal{M}, H, \mu_{t+1}) \end{aligned}$$

6.1 Output Sequence

The observable output of GödelOS is:

$$O_{\text{seq}} = \{O_0, O_1, \ldots, O_T\}.$$

7 Exception and Collapse Handling

The implementation must detect:

- Diverging stance
- Invalid tensor output
- Manifold projection errors

7.1 Recovery Rule

$$x_{t+1} = \text{Proj}_{\mathcal{M}}(x_t)$$

If unresolvable, the system halts with status *collapsed*.

8 Figures

Figure 1: GödelOS reference implementation flow.

9 Conclusion

This document formalizes the exact sequence of computations defining the GödelOS v0 execution architecture. Any implementation adhering to these semantics constitutes a valid GödelOS instance.