Abstract

Paper 1: Axis — A Semantic Substrate for AI-Native Computation

Title

Axis: A Semantic Substrate for AI-Native Computation

Author

Chris Taylor

Email: chris@axis-foundation.org

https://github.com/axis-foundation/axis-research

Abstract (Draft)

This paper introduces **Axis**, a semantic computation substrate designed specifically for AI systems. Unlike traditional programming languages, which evolved around human ergonomics and historical constraints, Axis is intended to provide a minimal, deterministic, formally defined logic language that allows AI models to express computation with clarity, stability, and verifiable semantics.

Axis addresses a growing industry problem: modern AI models generate code in Python, JavaScript, Rust, and other languages whose semantics were never designed for machine reasoning. These languages contain ambiguity, non-determinism, and historical irregularities that lead to hallucinations, inconsistency, and unreproducible behavior. Axis proposes a radical alternative: provide AI with a purpose-built semantic layer that is *stable*, *consistent*, *machine-verifiable*, and *language-agnostic*, while remaining interpretable by humans.

The core contributions of this paper are:

1. A description of the Axis paradigm: computation expressed through stable semantic structures instead of surface syntax.
2. A minimal deterministic language suitable as an AI reasoning target.

3. A unified conceptual model for reproducible, cross-language execution.
4. A discussion of the role of Axis as a long-term substrate for infrastructure, scientific computing, and autonomous systems.

This paper represents the conceptual foundation for a broader research program that includes formal semantics, immutable semantic contract catalogues, cross-language bridges, system-level semantics, and AI reasoning models.

---

1. Introduction

Modern AI systems have reached a point where they can generate complex software, reason about program structure, and transform code across languages. Yet the foundation they operate upon — human-designed programming languages — remains fundamentally misaligned with machine reasoning. These languages evolved through historically accumulated semantics, human ergonomics, and backwards compatibility constraints. They were *not* designed for deterministic symbolic reasoning, formal verification, or cross-language reproducibility.

As a result, AI-generated programs today suffer from well-known failure modes:

- inconsistent logic across iterations
- drift and mutation when rewriting code
- ambiguous or underspecified semantics
- hallucinated APIs and invalid transformations
- brittle behaviour due to runtime conditions AI cannot predict

This mismatch between **AI capability** and **language semantics** now defines the limits of AI-assisted software engineering.

Axis addresses this mismatch by introducing a new computational substrate: a **minimal, deterministic, AI-native logic language** paired with **a global, immutable semantic contract catalogue**. Instead of forcing AI to conform to programming languages with ambiguous semantics, Axis provides a stable and predictable semantic foundation for expressing computation.

Axis acts as a **semantic layer beneath all of the existing languages**: a canonical representation that AI systems use for reasoning and synthesis, with deterministic rules for translation into Python, Rust, JavaScript, C, Go, all the way to safety-critical stacks such as Ada/SPARK in aerospace, defence, and medical systems, where Axis's potentially provable semantics could, in principle, map into such systems if formal proofs of Axis semantics are completed in future work.

1.1 Motivation

LLMs have reached unprecedented capability but lack a stable representation of computational intent. They are forced to reason over:

- inconsistent semantics
- rapidly evolving libraries
- ambiguous APIs
- languages full of historical irregularities

Every AI-generated program is built atop shifting sands.

Axis supplies the missing foundation: a **semantics-first computational model** designed specifically for AI reasoning.

1.2 The Problem Space

AI-generated code is unstable because the substrate is unstable. Specifically:

1. **Programming languages were never designed for AI.** Human-centric syntax, ambiguous semantics, and accidental complexity confuse models.
2. **Libraries change unpredictably.** Functions disappear, rename, or refactor, invalidating training data.
3. **Cross-language translation is lossy.** Converting Python → Rust or JS → Go often introduces drift.
4. **There is no canonical IR for AI reasoning.** Each model improvises its own latent representation.
5. **There is no global semantic contract catalogue.** AI repeatedly relearns common operations from scratch across ecosystems.

Axis solves these issues with a single unifying framework.

1.3 Why Now? The AI Inflection Point

Axis emerges precisely when:

- AI-generated code volume is exploding
- hallucination and drift are becoming unacceptable
- enterprises demand reproducibility and auditability
- LLMs are powerful enough to benefit from formal semantics
- multi-language complexity is accelerating

Software is transitioning from human-authored to **AI co-authored**, and current languages cannot support this transition.

Axis provides the world's first computational substrate designed *for and with AI.*

---

2. What Axis Is

Axis is a **semantic computation substrate**, not a general-purpose programming language. It provides a minimal, deterministic representation of computation that is easy for AI to generate, reason about, and transform without ambiguity.

At its core:

> **Axis is a stable semantic language that AI uses to express computation, paired with deterministic rules for producing equivalent programs across host languages.**

Axis is built on three central pillars.

2.1 A Minimal Core Language

Axis contains a deliberately small set of constructs:

- clear, declarative expressions
- deterministic evaluation
- formally defined types
- no ambiguous runtime behaviour

The language's goal is not expressiveness but **semantic predictability**.

2.2 Deterministic Evaluation

Axis defines:

- a complete operational semantics
- deterministic memory behaviour
- predictable side-effects
- strict evaluation ordering

AI cannot hallucinate alternate interpretations because none exist.

2.3 The Global Semantic Catalogue

Axis is paired with a universal, immutable catalogue of semantic contracts. Each function:

- has a permanent ID
- is versioned without deletion
- cannot disappear from the ecosystem
- is cross-language consistent

This solves one of AI's biggest pain points: libraries that evolve unpredictably.

2.4 Cross-Language Reproducibility

Axis programs can target:

- Python
- Rust
- JavaScript/TypeScript
- Go
- C/C++
- future OS, network, and DB backends
- critical systems (e.g., Ada/SPARK), if future work succeeds in establishing a provably correct semantic foundation for Axis.

The same Axis program always evaluates to the same semantics, and depending on bridge implementations potentially identical output.

2.5 AI-Native Design

Axis is intentionally:

- small enough to learn fully
- deterministic enough to trust
- structured enough to avoid drift
- interpretable enough for collaboration

Axis is designed for **AI-first cognition**, not human ergonomics.

---

3. The Axis Paradigm Shift

Axis represents a shift as fundamental as the transition from assembly to high-level languages — but aimed at AI rather than humans. It reframes computation around **semantics**, not syntax, and provides a deterministic substrate that reduces or structurally prevents the most common causes of hallucination and drift inherent in traditional languages.

This paradigm shift can be understood in four layers:

3.1 From Syntax-First to Semantics-First

Traditional programming is syntax-led:

- syntax → semantics → execution

AI generation inverts this flow: models think semantically, then try to emit syntactically valid code after the fact.

Axis aligns with the *actual cognitive order* of AI:

- semantics → deterministic representation → host-language rendering

This alignment eliminates most forms of hallucination, drift, and inconsistent transformations.

3.2 From Ad-Hoc Libraries to a Stable Semantic Contract Catalogue

In today's ecosystems:

- functions get renamed
- APIs break
- semantics change
- models are trained on obsolete versions

Axis introduces the first-ever **immutable catalogue of semantic contracts** that describes how functions behave across ecosystems.

This stabilizes the entire computational universe for AI systems.

AI no longer guesses which version of a function exists — it knows.

Axis does not aim to replace existing software ecosystems. Frameworks, libraries, and language communities remain where they are.

Instead, Axis replaces the semantic core — the low-level functional primitives (such as numeric operations, transforms, data structure operations, and deterministic algorithms) where ambiguity and cross-language drift produce inconsistent behaviour for AI systems.

By standardising this foundational layer, existing libraries can continue to evolve normally while AI systems gain a stable, verifiable substrate for reasoning and generation.

3.3 From Human-Centric Languages to AI-Native Languages

Human languages evolved for:

- readability
- ergonomics
- familiarity
- tooling constraints

Axis evolves for:

- deterministic semantics
- minimal surface area
- machine reasoning
- translation invariance
- formal verification

AI finally gets a language designed for its internal logic, not ours.

3.4 From Unpredictable Execution to Deterministic Evaluation

Axis removes:

- undefined behaviour
- order-of-evaluation ambiguity
- runtime irregularities
- hidden memory effects

The result is:

- predictable evaluation
- reproducible transformations
- trustworthy reasoning processes

This enables a pathway towards potentially **provable AI code generation**.

3.5 From Fragmented Ecosystems to a Unified Semantic Substrate

Today:

- Python, Rust, JS, C++ each encode different worldviews
- AI must learn all of them
- each change in each ecosystem creates drift

Axis provides **one unified semantic layer** underneath all languages.

Host languages become *rendering targets*, not reasoning substrates.

This changes the nature of software development permanently:

- AI writes semantics
- Axis creates the semantic conditions under which correctness guarantees become possible
- bridges generate host-language code

Axis becomes a semantic source of truth for computation.

4. High-Level Architecture

Axis introduces a layered semantic architecture that cleanly separates *intent*, *semantics*, *implementation*, and *execution*. This separation is fundamental to ensuring determinism, stability, and cross-language reproducibility. The architecture is intentionally minimal, yet expressive enough to serve as a universal substrate across infrastructure, languages, systems, and AI reasoning processes.

Axis organizes computation into **three primary planes**, each with a distinct purpose:

1. **The Semantic Plane** — the Axis core language
2. **The Contract Plane** — immutable semantic contract catalogue
3. **The Execution Plane** — host-language backends

Together, these planes define a complete pipeline from **AI intent → deterministic semantics → executable program**.

4.1 The Semantic Plane: Axis Core Language

The semantic plane contains the minimal set of constructs that define computation within Axis.

It includes:

- formal syntax

- type system
- static semantics (typing rules)
- dynamic semantics (evaluation rules)
- deterministic memory model

This layer is the *single source of truth* for computational behaviour. No ambiguity, no hidden effects, and no undefined behaviours exist at this level.

The purpose of the semantic plane is to ensure that:

- the same Axis program always evaluates the same way,
- AI systems have a stable representation for reasoning,
- cross-language backends cannot introduce drift.

This is the conceptual heart of Axis.

4.2 The Contract Plane: Federated, Immutable Semantic Catalogue

Above the core semantics, Axis introduces a **federated contract plane** — a distributed catalogue of permanent, versioned semantic contracts describing the behaviour of core computational primitives. Each contract:

- has a globally unique, namespaced ID
- evolves only through **additive**, backward-compatible versioning
- is immutable once published
- defines behaviour in deterministic, language-agnostic terms
- may have implementations in multiple languages and environments

Unlike traditional libraries, which change over time and diverge across ecosystems, Axis contracts provide **stable, long-lived semantic definitions** for the operations that matter most to AI reasoning — numeric primitives, structural transforms, algorithmic kernels, and other foundational behaviours.

Axis does **not** replace existing ecosystems. Instead, it offers a shared semantic foundation beneath them:

- core behaviours become consistent across languages,
- semantic drift is reduced,
- AI systems gain a reliable reference point independent of library evolution.

Over time, this contract layer enables a **deterministic, potentially provable computational substrate** that is robust to changes in programming languages, packages, and toolchains.

4.3 The Execution Plane: Multi-Language Backends

Below the semantic plane sit host-language backends that evaluate Axis programs. These include:

- Python
- JavaScript / TypeScript
- Rust
- Go
- C / C++
- (future) OS-level semantics
- (future) network and infrastructure semantics

Each backend:

- implements the Axis operational semantics,
- maps Axis types into host equivalents,
- ensures deterministic evaluation,
- provides bridge logic for contract invocations.

Host languages are **not** responsible for defining meaning — only for **executing** the deterministic meaning defined in Axis.

The execution plane therefore becomes replaceable, optimizable, and independently verifiable.

4.4 Architectural Relationships

The relationship between these three planes can be expressed succinctly:

- **Axis Core** provides *semantics*
- **Catalogue** provides *stable functional vocabulary*
- **Backends** provide *execution*

All three together create a pipeline:

AI intent → Axis semantics → contract resolution → backend execution

This pipeline is deterministic, composable, and reproducible across languages.

4.5 Diagram: Three-Plane Axis Architecture

```
AXIS SEMANTIC PROCESSING PIPELINE

AI Intent / Human Collaboration
   |
   v

Axis Core Language (Semantic Plane)
   |
   v

Immutable Semantic Catalogue (Contract Plane)
   |
   v

Backend Execution
(Python / Rust / JavaScript / Go / C / OS / Network)
   |
   v

Deterministic Program Output

Intent -> Semantics -> Contract Resolution -> Execution -> Output
```

4.6 Why This Architecture Works

This separation achieves several critical goals:

Determinism

Semantics are defined once, globally, without deviation across backends.

Stability

Contracts never disappear, so AI systems can rely on them indefinitely.

Composability

Axis programs can combine semantics and registry functions without concerning themselves with backend details.

Portability

The same Axis program can be executed in any supported backend with identical results.

AI-Optimal Reasoning

By removing syntactic and semantic noise, Axis becomes the ideal medium for machine reasoning, optimisation, and transformation.

Long-Term Extensibility

The architecture naturally extends to OS-level semantics, networking, filesystems, and distributed systems — without semantic drift.

Axis is not just another language. It is a **semantic operating layer** for computation itself.

---

5. Applications and Impact

Axis is not designed as a niche programming language or theoretical construct. It is intended as a **unifying semantic substrate** — a foundation for deterministic computation in environments where reliability, reproducibility, and semantic clarity are essential. Because Axis decouples *meaning* from *execution*, its impact extends across multiple layers of modern computing.

This section highlights several domains where Axis provides immediate and transformative advantages.

5.1 AI Code Generation and Transformation

The most immediate application of Axis is in AI-driven software development. Current LLMs generate code by navigating ambiguous languages with irregular semantics, which leads to:

- hallucinated APIs
- inconsistent behaviour across iterations
- difficulty maintaining state across transformations
- unpredictable cross-language translations

Axis removes these failure modes by giving AI a predictable semantic target.

AI models benefit from Axis in three ways:

1. **Reduced Cognitive Ambiguity** The deterministic semantics eliminate multiple interpretations of the same program.
2. **Stable Function Vocabulary** The immutable registry provides a consistent set of operations across all languages.
3. **Reproducible Transformations** Axis → Python → Axis → Rust round-trips are intended to be lossless under deterministic backends, avoiding drift.

This establishes Axis as the *lingua franca* for AI-native reasoning about code.

5.2 Scientific and Numerical Computing

Scientific computing demands:

- reproducibility
- determinism
- precise control over numerical semantics
- stable, long-lived computations

Axis ensures:

- consistent evaluations across hardware and languages
- permanent, verifiable definitions of numerical operators
- transparent and auditable computational pipelines
- cross-language equivalence of scientific routines (Python, Rust, C, etc.)

This eliminates a multi-decade pain point where scientific libraries evolve and silently break old models.

Axis stabilizes scientific software ecosystems at the semantic level.

5.3 Enterprise and Infrastructure Systems

Enterprise systems—financial platforms, logistics engines, authentication systems, large-scale ETL, regulated workloads—suffer from instability arising from:

- dependency drift
- versioning conflicts
- language/runtime migration
- developer turnover
- inconsistent implementations across services

Axis introduces a foundation where:

- semantics remain constant even as implementations change
- policies and logic can be expressed once and executed anywhere
- cross-language systems share a unified operational behaviour
- drift and regression risks fall dramatically

This makes Axis especially valuable for:

- banks
- medical systems
- compliance-heavy industries
- large multi-team enterprises

In these domains, determinism is not optional; it is essential.

5.4 Autonomous Systems

Robotics, autonomous vehicles, drones, and decision-making agents must operate with:

- predictable logic
- verifiable transitions
- formal guarantees
- consistent outcomes across environments

Axis provides:

- deterministic state transitions
- formalizable operational semantics
- long-lived, versioned behaviour definitions

Axis also makes verification more tractable, because semantics are deterministic and formally definable.

It becomes possible to specify control logic once and trust it across hardware generations, languages, and platforms.

5.5 Multi-Language Ecosystems and Migration

Modern organizations operate large portfolios of software written in:

- Python
- JavaScript
- Rust
- Go
- Java
- C++

Historically, migrating between these languages is error-prone and often impossible without drift.

Axis enables:

- lossless semantic translation
- long-lived logic expressed independent of implementation
- incremental migration of legacy systems into safer forms

This finally brings convergence to ecosystems that have been fragmenting for decades.

5.6 Towards Semantic Infrastructure

Axis lays the groundwork for something unprecedented: **formal semantics for infrastructure itself**.

Future layers (explored in further papers) will allow deterministic reasoning about:

- network routing
- firewall rules
- filesystem and process semantics
- orchestration policies
- database queries
- distributed protocols

This collapses the gap between *software* and *infrastructure*, both expressed in a shared semantic substrate.

5.7 Summary: Why Axis Matters

Axis impacts computing in four deeply structural ways:

1. **It gives AI a stable language for thought.**
2. **It unifies computation across languages, platforms, and ecosystems.**
3. **It defines deterministic semantics across environments that currently lack it.**
4. **It expands semantics beyond code into infrastructure and system design.**

Axis is not simply a new language but a foundational shift in how computation is described, reasoned about, and executed.

---

6. Conclusion

Axis proposes a new foundation for computation in the era of AI. It begins from a simple observation: modern programming languages, designed for human ergonomics and historical

constraints, are no longer adequate as substrates for AI reasoning. Their ambiguity, irregularity, and semantic drift introduce structural instability into every AI-generated program.

Axis offers a fundamentally different approach:

- **a minimal, deterministic semantic language**,
- **a global immutable semantic catalogue**, and
- **a set of cross-language backends** that faithfully implement the same semantics everywhere.

This architecture forms a stable computational substrate — one that enables reproducible, verifiable AI-generated code across languages, systems, and domains.

The significance of this shift is both immediate and long-term:

- AI systems benefit from a cleaner, more interpretable reasoning space.
- Enterprises gain stronger guarantees around reproducibility and stability.
- Scientific computing acquires a deterministic cross-language foundation.
- Infrastructure and distributed systems become amenable to formal semantics.
- Autonomous systems gain a predictable, verifiable operational core.

Axis does not compete with existing languages; it **underlies them**. It does not replace human programming; it **augments it** through a semantics-first model that removes ambiguity and drift. And it does not attempt to dictate how software should be written; it defines how meaning should be expressed so that AI systems can reason about it clearly.

The ideas in this paper represent the starting point of a broader research program that will unfold through additional work on formal semantics, registry governance, bridging architectures, system-level semantics, and AI-native reasoning models. Together, these components outline a future where computation is defined not by syntax, but by stable, precise, and universally shared semantics.

Axis is a step toward that future — a future where AI and humans collaborate on top of a deterministic, extensible, and rigorously defined semantic substrate that supports the next generation of software systems.

References

Transformers / AI Reasoning

- Vaswani, A. et al. (2017). *Attention Is All You Need.*

Formal Semantics / Programming Languages

- Pierce, B. C. (2002). *Types and Programming Languages.*
- Plotkin, G. (1981). *A Structural Approach to Operational Semantics.*
- Milner, R., Tofte, M., Harper, R., & MacQueen, D. (1997). *The Definition of Standard ML.*

Formal Correctness / Deterministic Systems

- Hoare, C. A. R. (1969). *An Axiomatic Basis for Computer Programming.*
- Lamport, L. (1998). *The Part-Time Parliament.* (optional)

Appendix A — Glossary

This glossary defines key terms used throughout the paper. Many of these terms anchor conceptual elements that will be expanded in subsequent papers in the Axis series.

Axis

A semantic computation substrate designed for AI-native reasoning, consisting of a minimal deterministic language, a global semantic contract catalogue, and multi-language execution backends.

Semantic Plane

The layer of Axis that defines the core language's syntax, type system, and evaluation rules. It is the source of truth for computational meaning.

Contract Plane

The global registry of immutable function contracts. Each function has a permanent identifier, version history, and deterministic semantics.

Execution Plane

The collection of backends (Python, Rust, JavaScript, Go, C, OS-level semantics, etc.) that implement the formal semantics defined in Axis.

Contract

A permanent function definition in the registry. Contracts define semantic behaviour independently of any host-language implementation and cannot be removed once published.

Versioning (Contract Evolution)

The process of adding new versions to a contract while preserving all previous versions. Axis prohibits deletion or mutation of past definitions, ensuring temporal stability.

Semantic Drift

The phenomenon where changes in code, libraries, or languages cause the meaning of a program to diverge over time. Axis prevents this through deterministic semantics and immutable contracts.

Deterministic Evaluation

The guarantee that an Axis program always evaluates the same way in every execution context, backend, and runtime environment.

Semantic Substrate

A foundational layer beneath programming languages that defines meaning independently of syntax or runtime environment.

Cross-Language Equivalence

The guarantee that an Axis program translated to multiple host languages (e.g., Python, Rust, JS) produces identical behaviour.

LLM Reasoning Substrate

Axis provides a stable representational format that language models can reliably learn, manipulate, and transform.

Law Tests

Small reference programs used to validate the correctness and consistency of Axis evaluation rules across implementations.

Host-Language Bridge

A deterministic mapping layer that translates Axis semantics into executable constructs for a specific programming language.

Semantic Intent

The abstract computational meaning expressed by an Axis program, independent of syntax or implementation.

Appendix B — Figures and Diagrams

This appendix provides locations for diagrams referenced in the paper. Diagrams will use Mermaid syntax unless otherwise noted. These diagrams visually anchor the conceptual and architectural elements of Axis.

B.1 Axis Three-Plane Architecture

```
AXIS THREE-PLANE MODEL

[SEMANTIC PLANE]
  - Axis Core Language
  - Types, Static Semantics
  - Evaluation Rules
  - Deterministic Memory Model
        |
        v
[CONTRACT PLANE]
  - Immutable Function Registry
  - Permanent Versioned Contracts
  - Stable Behaviour Definitions
        |
        v
[EXECUTION PLANE]
  - Python
  - Rust
  - JavaScript / TypeScript
  - Go
  - C / C++
  - OS / Network (future)
```

Meaning is defined once (Semantic Plane), vocabulary is stabilised (Contract Plane), execution is backend-specific but deterministic (Execution Plane).

Description: This diagram illustrates the relationship between Axis's three fundamental layers. Semantics define meaning, the registry defines functional vocabulary, and backends define execution.

B.2 AI-to-Axis Reasoning Pipeline

```
AI -> AXIS REASONING PIPELINE

AI System (LLM)
        |
```

```
        v
Semantic Intent
        |
        v
Axis Representation
        |
        v
Contract Resolution (Registry)
        |
        v
Backend Execution
(Python / Rust / JS / Go / OS)
        |
        v
Deterministic Output
```

Description: Shows how AI intent flows through Axis semantics and contract resolution into deterministic host-language execution.

B.3 Cross-Language Execution Model

```
CROSS-LANGUAGE EXECUTION MODEL

            Axis Program
       /         |          |          \
      v          v          v           v
   Python      Rust     JavaScript     Go
      |          |          |           |
      v          v          v           v
   Deterministic Output (Same Meaning)
```

Description: Demonstrates that all host backends potentially yield identical results from the same Axis program.

B.4 Semantic Drift vs Deterministic Semantics

```
TRADITIONAL SYSTEMS vs AXIS SEMANTIC MODEL

Traditional Languages
    |
    +-- Libraries change
    +-- APIs break
    +-- Behaviour drifts
    +-- Old code becomes unreliable
```

```
Axis Model
    |
    +-- Contracts are immutable
    +-- Semantics are permanent
    +-- Behaviour is reproducible
    +-- Programs are stable long-term
```

Description: Highlights how Axis could potentially eliminate semantic drift over time.

B.5 Axis Future Extension Layers (Concept Diagram)

```
AXIS FUTURE EXTENSION LAYERS

Axis Core Language
        |
        v
Function Registry
        |
        v
Bridge Layer
   |------- Axis Web Semantics
   |------- Axis OS Semantics
   |------- Axis Network Semantics
   |------- Axis Database Semantics
   |------- Axis Reasoning Layer
```

Description: Illustrates how Axis extends upward into higher-order system semantics and downward into AI reasoning models.