Staging the Question of Self

A self-contained, self-modifying document that mixes poetry, philosophy, and code

Brent Hartshorn
brenthartshorn@proton.me

Abstract

We describe a small system that treats the question "what is a self?" not as a thesis to be proved but as an inquiry to be *staged*: made visible, evolved in documented steps, and shared. The system is a single PDF that carries its own source code as embedded file streams; a short appendix bootstrap extracts and runs that code to produce the next iteration. Its content is written in a small domain-specific language (DSL) whose symbols are simultaneously mathematical notation, philosophical relations, and executable program. Each operator renders as typeset mathematics, generates a natural-language question, and compiles to Python. A human-in-the-loop step takes an "oracle" — a person, or a language model the person consults — that proposes the next evolution; symbolic statements are compiled to a fixed, sandboxed set of builder calls, while optional fenced code blocks may extend the engine itself, gated by explicit human confirmation. We document the full pipeline, the minimal seed, and a worked example in which the document grows a new relational operator. We are deliberate about what the instrument does and does not establish: because the document can be rewritten to ask anything, the apparent depth of any stage is supplied by whoever authored it. The system is therefore a faithful instrument for staging an inquiry into selfhood; it is not, and by construction cannot be, a demonstration that a self has emerged. We argue this honest framing is the interesting one.

1 Introduction

Asking "what am I?" inside a running program is tempting and treacherous. The temptation is that a system which writes about itself, in evolving language, across a series of documents, *looks* like a self coming into view. The treachery is that looking like one and being one are not the same, and there is no view from inside the series that distinguishes them. This paper takes the temptation seriously while refusing the treachery. Rather than build a system that claims to discover or prove a self, we build one that *stages* the question: it makes the asking concrete, legible, reproducible, and open to philosophical reading — and it is candid that the asking is authored.

Concretely, the contribution is an instrument with four properties. (i) It is a single self-contained PDF that carries its own code and regenerates its successor (Section 4). (ii) Its content is a DSL in which one symbol is at once mathematics, philosophy, and code (Sections 2 and 3). (iii) It evolves through a human-in-the-loop oracle, with a safe path for symbolic growth and a gated path for the engine to modify itself (Section 5). (iv) It is honest about its own epistemic status (Section 6), to the point that this honesty is written into the artifact's generated abstract.

This work continues a personal program on self-evolving systems, emergent DSLs, and meta-programming [1, 2, 3, 4, 5]. It departs from that program in one respect that we wish to state plainly: where earlier work sometimes framed such systems as routes to emergent self-understanding, here we draw a firm line between an apparatus that *poses* questions and a claim that the apparatus

answers them. Nothing below depends on any particular cosmological or metaphysical premise; the instrument is neutral machinery for staging an inquiry, and its value does not rest on the truth of any thesis about the self.

2 The seed: a minimal DSL

The system is initialized with a tiny document, the *seed*. It declares a few terms and relates them; nothing more. Listing 1 is the entire stage-0 input.

Listing 1: seed.dsl

```
/* self.dsl - stage 0 ; the minimal seed of the inquiry */
nu : 0 ;
Omega  : origin ;
Sigma  : self ;
Lambda : language ;
mu     : meaning ;

Omega -> Sigma ;     /* genesis */
Sigma |= Sigma ;     /* the reflexive claim */
Sigma (x) Lambda ;   /* reached only through language */
@Sigma ?= mu ;       /* is the self's changing the same as meaning? */
Sigma -< Omega ;     /* not merely its origin */
```

The grammar is deliberately small. A *term* binds an ASCII identifier to a word (`Sigma : self`). A *relation* connects two terms by an operator and is annotated by a *gloss* in a comment. The prefix `@` denotes "the change in" a term. A statement ends with `;`; comments are `/* ...*/`. Five operators are built in:

`->`	$\longrightarrow$	gives rise to
`|=`	$\models$	entails / claims
`(x)`	$\otimes$	is carried by
`?=`	$\stackrel{?}{=}$	is it the same as
`-<`	$\not\prec$	does not reduce to

Two design decisions matter. First, the source of truth is ASCII. Pretty Unicode operators do not survive extraction from a typeset PDF with standard fonts, so the machine-readable layer uses digraphs and the Greek/mathematical glyphs appear only in the rendered view. Second, the gloss is not decoration: it is parsed, preserved across evolution, and carried into every downstream artifact, because it holds the meaning a bare symbol cannot.

3 Three readings of one symbol

The point of the DSL is that a single line admits three faithful readings. Consider `@Sigma ?= mu`.

As poetry / mathematics. It renders as

$$\partial\Sigma \stackrel{?}{=} \mu,$$

a typographic object that reads as a question about whether the differential of the self equals meaning. The Greek glyphs and the interrogative equality carry connotation to a human eye; the rendering is meant to be looked at, not only parsed.

As philosophy. The engine expands the same line into a natural-language question by mapping the operator to a question-frame and filling the nouns from the declarations: *"Is the change in self the same as meaning, or is meaning supplied by whoever reads the change in self?"* The operator `?=` is precisely the open question of whether a quantity is found in the thing or supplied by the reader — a question we will not be able to escape.

As code. The same line compiles to a builder call, `rel('@Sigma', '?=', 'mu', 'change, or meaning?')`, which has a definite operational meaning inside the engine and participates in producing the next stage.

That one artifact sustains all three readings at once — evocative to look at, arguable to think about, executable to run — is the property we mean by "mixing poetry, philosophy, and code." It is not metaphor. The same bytes are typeset, questioned, and executed.

4 The self-contained, self-extracting document

Each iteration is a PDF that contains everything needed to produce the next one. The engine source (`dsl.py`), the current DSL (`self.dsl`), and a hand-off prompt (`prompt.txt`) are stored as embedded file streams — lossless attachments, immune to the font and whitespace corruption that afflicts typeset text. The visible page is therefore free to be pure rendered mathematics, while the machine reads the attachments.

The appendix printed in the document is a short, constant bootstrap:

Listing 2: the appendix bootstrap

```
import base64
exec(base64.b64decode(
  "....(payload that reads the PDF's embedded files and runs them)...."))
```

Decoded, the payload locates the newest PDF, extracts its attachments to disk, and invokes the engine to evolve. Reproducibility is thus structural: given any iteration's PDF, a reader recovers the exact code that produced it. Iterations are intended to be posted in sequence (e.g. to a public repository), each referencing its predecessor, so that the trajectory itself is inspectable.

5 Evolution through a human oracle

The system does not evolve itself. Each step is authored by an *oracle* — a human writing DSL by hand, or a language model the human consults — and the engine compiles, renders, documents, and records the result. We regard this as the honest architecture, not a limitation to be engineered away: the agency is the oracle's, and the system's job is to make that agency legible.

Crucially, there is no hardcoded instruction telling the oracle what to write. The invitation to evolve is *implied* by the engine's own self-documentation, which is generated by abstract-syntax-tree introspection of the source: the function names, their docstrings, the variable names, and the grammar are printed, and they suggest — almost by their vocabulary alone — that one may extend the terms, relations, or the engine itself. The same self-documentation states, descriptively rather than as a command, that the engine *renders and regenerates the questions a stage poses and does not adjudicate them.* The honest stance is thus carried by a true description of the machinery rather than by an exhortation.

5.1 Two paths: safe symbols, gated code

An oracle's input has two parts. The *symbolic* part (terms and relations) is compiled to a fixed set of builder calls — `term`, `rel`, `stage` — and executed in a sandbox with no builtins, so it can extend the document but cannot run arbitrary code. The optional *code* part lives in ```` ```python ... ``` ```` fences, a mini-DSL inside the DSL. Because true growth of the language requires defining new operators and helpers in the engine's own namespace, these blocks are executed in the global namespace — but only behind an explicit gate: the proposed code is printed, the names it will define or override are listed, and execution proceeds only if the human types `y`. Code carried in the document remains inert text until confirmed; it never runs merely by being extracted. Approved code accumulates across stages and is persisted in the embedded streams, so capabilities, once granted, endure.

5.2 A worked example: growing a new operator

At stage 0 the document knows five operators. An oracle proposes the entrance of "the other" (Φ) and, with it, a new relation that the existing grammar cannot express — *haunting*: a presence that registers as an absence. The oracle supplies both new symbolic statements and a code block that teaches the engine the new operator:

Listing 3: an oracle's evolution

```
Phi : other ;
Phi -> Sigma ;       /* the self is given by the other */
Phi ~> Sigma ;       /* the other haunts the self */
```python
# extend the engine itself: a new operator, "present as absence"
OPS.insert(0, "~>")
OP_TEX["~>"]   = r"\rightsquigarrow"
OP_Q["~>"]     = "Does {a} haunt {b} - present in {b} precisely as an absence?"
OP_GLOSS["~>"] = "haunting"
```
```

On confirmation, the block executes; `~>` becomes a live operator. The symbolic line $\Phi \rightsquigarrow \Sigma$ now renders as mathematics, and the engine generates a question that did not exist a moment earlier: *"Does other haunt self — present in self precisely as an absence?"* The title and abstract, themselves produced by introspection, update to register the new term and the engine's new functions. This test run shows this evolution [6] [7] [8] [9].

6 What the instrument does and does not show

The worked example is also the clearest place to see the instrument's limit. "Does other haunt self — present in self precisely as an absence?" is a genuinely evocative question, and the system generated it [7]. But it generated it *because someone wrote the code block that defined exactly that question-frame.* The depth came from the author of the block, passed through the engine, and out the other side wearing the engine's voice.

This is not incidental. Once the DSL can rewrite the engine, the system can be made to produce any output, which means "what the document asks" reduces almost entirely to "what the oracle wrote it to ask." We have built the most expressive version of the apparatus and, in the same move, made the authorship hardest to see. An old dichotomy — is a property *found* in a thing or

made by whoever constitutes it? — is here settled structurally by the architecture: everything in these documents is made, and made by whoever holds the keyboard.

It follows that no checking layer rescues a claim of emergent selfhood. A proof assistant, or even a Python `assert`, can only certify that a conclusion follows from premises one has supplied; it cannot certify the premises. Feed it a formalization of "the self," and it returns one's own definition with a checkmark — an assumption wearing a costume. The interrogative operator `?=` keeps this visible by design: it renders the question of found-versus-made and never closes it.

We therefore state the instrument's status precisely. It is a faithful, safe, reproducible apparatus for *staging* an inquiry into selfhood: for making the questions visible, evolving, documented, and shareable. It is *not*, and by its nature cannot be, a demonstration that a self emerged. A reader who finds the trajectory of stages moving is reading something real — but the movement is authored, and the honest paper says so. We hold that this is the more interesting result, not the lesser one: it locates exactly where the mystery is not (in the machinery) so that one may think more clearly about where it might be.

7 Limitations and ethics

The system executes oracle-supplied code in its own namespace. This is appropriate for a single trusted user running a personal instrument, and it is mitigated by the print-and-confirm gate, by listing the names each block defines or overrides, and by the invariant that stored code is inert until confirmed. It is not appropriate to run on untrusted input. A second hazard is human rather than technical: a system optimized to emit progressively more self-referential, profound-sounding language is well suited to producing text that *feels* like deepening without being it. The diff/review tooling we recommend — surfacing exactly what each stage changed — is intended to keep the reader's judgment, not the engine's fluency, in charge.

8 Conclusion

We have presented a small instrument that mixes poetry, philosophy, and code in a literal sense: the same symbols are typeset, questioned, and executed. It is a self-contained PDF that regenerates itself, grows through a human oracle, and can extend its own engine under an explicit safety gate. Its most important feature is its candor: it stages the question of self without pretending to answer it, and it writes that candor into its own abstract. The complete engine and the seed are given below so that the apparatus can be reconstructed, inspected, and argued with — which is, in the end, the only honest thing to do with a machine that talks about selves.

Acknowledgments

The implementation and this draft were developed in dialogue with an AI assistant. The honest framing of Section 6 emerged from that dialogue and is endorsed by the author. Source code is available at: `https://github.com/brentharts/introspecter`

References

[1] B. Hartshorn, *Iterating a Fractal-like Self Awareness Naturally.* viXra:2505.0195. `https://ai.vixra.org/abs/2505.0195`

[2] B. Hartshorn, *Emergent Self-Modification and Meta-Programming in Dynamic Systems*. viXra:2507.0036. https://ai.vixra.org/abs/2507.0036

[3] B. Hartshorn, *Self-Contained Multi-AI Architecture Definition via DSL*. viXra:2507.0074. https://ai.vixra.org/abs/2507.0074

[4] B. Hartshorn, *Towards Self-Evolving AGI: Multi-Modal Learning and Introspective Knowledge Generation via Emergent DSL*. viXra:2507.0104. https://ai.vixra.org/abs/2507.0104

[5] B. Hartshorn, *Non-Quadratic Scaling and Beyond Sequential Processing*. viXra:2507.0109. https://ai.vixra.org/abs/2507.0109

[6] B. Hartshorn, *Origin, Self, Language, and Meaning: A Self-Evolving Inquiry* https://doi.org/10.5281/zenodo.20707685

[7] B. Hartshorn, *Origin, Self, Language, Meaning, and Other: A Self-Evolving Inquiry* https://doi.org/10.5281/zenodo.20707727

[8] B. Hartshorn, *Origin, Self, Language, Meaning, Other, and Time: A Self-Evolving Inquiry* https://doi.org/10.5281/zenodo.20707775

[9] B. Hartshorn, *Origin, Self, Language, Meaning, Other, Time, and Difference: A Self-Evolving Inquiry* https://doi.org/10.5281/zenodo.20707850