

# THE ZERO PARADOX

## ZP-K: Computational Grounding of Self-Reference

Version 1.2 | April 2026

v1.2: DA-1 Path 2 recharacterized — foundational commitment, not missing proof. Forward paths: new axiom, Chalmers' implementation, or ZP-PQ dissolution. DA-1 does not depend on Path 2. | v1.1: Remark R-K.0 added — T-COMP four-way equivalence clarified: (1)–(3) equivalent by T-EXEC (derived); (4) combined by KleeneStructure typeclass requirement (structural commitment, not independent derivation). | v1.0: Four-way equivalence proved — Quine atom =  $\perp$  = join identity = Kleene fixed point. KleeneStructure typeclass bridges AFA self-containment to Kleene's second recursion theorem. DA-1 formally closed: `da1_closed_concrete : IsQuineAtom( $\perp$  : MachinePhase)`. All ZPK.lean theorems verified in Lean 4.

---

This document establishes the computational grounding of the Zero Paradox's central self-reference structure. The key insight (April 2026):  $\perp$  in the computational instantiation is not a state of a Turing machine.  $\perp$  IS the universal Turing machine in its ground state — the executor for which no external executor exists. Kleene's second recursion theorem (Mathlib: `Nat.Partrec.Code.fixed_point2`) provides the formal witness: a code that IS its own program, the computational expression of  $\perp = \{\perp\}$ .

The central result is a four-way equivalence. The structural roles of  $\perp$  — Quine atom (set-theoretic), bottom element (order-theoretic), join identity (algebraic), and Kleene fixed point (computational) — are not analogies. They name the same structural object in four formal languages. The bridge from mathematical self-reference to computational execution is not a bridge. It is a recognition of identity.

---

## Section I: The Kleene Fixed Point

### I. Kleene's Second Recursion Theorem

Kleene's second recursion theorem is the computational fixed-point theorem. For any partially computable transformation  $f$  of programs, there exists a program  $e$  such that  $e$  and  $f(e)$  compute the same function. Applied to the identity: there exists a program that computes the same function as itself — a program that is its own program.

In Lean 4, this is `Nat.Partrec.Code.fixed_point2` in Mathlib's computability library. For any `Partrec2` function  $f$  (a partially computable transformation of codes), there exists a `Code`  $c$  such that `eval c = f c`. The existence is non-constructive (`Classical.choice`), which is why all ZP-K theorems carry the Mathlib axioms [`propext`, `Classical.choice`, `Quot.sound`].

#### Kleene's Second Recursion Theorem (Mathlib: `fixed_point2`)

`Nat.Partrec.Code.fixed_point2`: For any partially computable  $f : \text{Code} \rightarrow \mathbb{N} \rightarrow \cdot \mathbb{N}$ , there exists  $c : \text{Code}$  such that `eval c = f c`.

## Kleene's Second Recursion Theorem (Mathlib: fixed\_point<sub>2</sub>)

This is the computational expression of the Quine atom. A code whose behavior is determined by itself alone — no external description shorter than  $c$  generates it. The computational analogue of  $\perp = \{\perp\}$ .

## II. The Self-Application Map

The self-application map sends each code  $c$  to the partial function that runs  $c$  on  $c$ 's own Gödel number plus an offset. A fixed point of self-application is a code that computes its own behavior — running it on any input gives the same result as running it on its own encoding plus that input.

### Definition: selfApply and IsComputationalQuine (ZPK.lean § I)

`selfApply : Code → ℕ → ℕ := fun c n ↦ eval c (encode c + n)`

A code  $c$  is a computational Quine if `eval c = selfApply c`, i.e.:

$\forall n, \text{eval } c \ n = \text{eval } c \ (\text{encode } c + n)$

This is a periodicity condition:  $c$ 's output at  $n$  equals its output at  $\text{encode}(c) + n$ . The encoding plays the role of the "address" of the program —  $c$ 's behavior at  $n$  is the same as  $c$ 's behavior at its own address plus  $n$ .

`selfApply_partrec`: `selfApply` is partially computable.

Proof: `eval_part` (Mathlib) composed with `Primrec.encode` and `Primrec.nat_add`. Lean purity: [`propext`, `Classical.choice`, `Quot.sound`]. ✓

### Theorem: computational\_quine\_exists

There exists a computational Quine:  $\exists c : \text{Code}, \text{IsComputationalQuine } c$ .

Proof: immediate from `kleene_fixed_point_exists` applied to `selfApply`, using `selfApply_partrec`.

Lean purity: [`propext`, `Classical.choice`, `Quot.sound`]. ✓

Note on uniqueness: unlike the AFA Quine atom (unique by the AFA decoration theorem), computational Quines are not unique. Multiple codes can satisfy the fixed-point equation independently. Uniqueness in ZP-K flows from ZP-J T-EXEC (on the set-theoretic side), not from the computational definition.

## Section II: KleeneStructure — Bridging AFA and Computation

### I. The Bridge

ZP-J's `AFAStructure` typeclass encodes AFA self-containment in type theory: a predicate `selfMem`, AFA uniqueness (`quine_unique`), and the bridge field `bot_self_mem` ( $\perp$  is self-containing). T-EXEC follows: the Quine atom equals  $\perp$ .

ZP-K's `KleeneStructure` extends `AFAStructure` with a computational witness: a code `botCode` that IS its own program, whose existence is guaranteed by Kleene's theorem. The AFA self-containment ( $\perp = \{\perp\}$ ) and the

Kleene fixed point (botCode is its own program) are the same structural fact stated in two formal languages.

### KleeneStructure Typeclass (ZPK.lean § II)

class KleeneStructure (L : Type\*) [ZPSemilattice L] extends AFAStructure L with:

(inherited) selfMem : L → Prop — self-membership predicate

(inherited) quine\_unique — AFA uniqueness

(inherited) bot\_self\_mem —  $\perp$  is self-containing

(new) botCode : Code — the code witnessing  $\perp$ 's computational self-reference

(new) botCode\_is\_quine : IsComputationalQuine botCode — botCode IS its own program

(new) bot\_self\_mem\_from\_kleene : selfMem  $\perp$  — the Kleene side implies the AFA side

Any KleeneStructure instance must supply both the AFA witness (bot\_self\_mem) and the computational witness (botCode with botCode\_is\_quine). The two are required together because they are the same structural fact.

## II. Why Not a Bridge Axiom?

The identification between AFA self-containment and Kleene computational fixed points is not asserted as a new axiom. KleeneStructure is a typeclass: any concrete type claiming this identification must discharge it as a proof obligation. The commitment is checked at instantiation, not accepted globally.

The distinction from ZP-J carries over: a freestanding axiom says "trust me." A typeclass field says "prove it for your specific type, or it does not compile." The MachinePhase instance in § V shows how the obligation is discharged concretely.

## Section III: T-COMP — The Four-Way Equivalence

ZP-J T-EXEC established a three-way equivalence: Quine atom (set-theoretic)  $\leftrightarrow$  bottom element (order-theoretic)  $\leftrightarrow$  join identity (algebraic). ZP-K adds the fourth: Kleene fixed point (computational). The four characterisations of  $\perp$  are present simultaneously in any KleeneStructure lattice.

### Remark R-K.0 — What "Four-Way Equivalence" Means

The equivalence among (1)–(4) has two distinct sources:

(1)–(3) are equivalent by T-EXEC (ZP-J): any element that is a Quine atom is also  $\perp$  and a join identity, and vice versa. This is a genuine logical derivation — the three properties are proved to coincide from the AFAStructure axioms.

### Remark R-K.0 — What "Four-Way Equivalence" Means

(4) is present in any KleeneStructure instance because KleeneStructure requires it as a typeclass field: botCode\_is\_quine must be supplied at instantiation. There is no independent proof that satisfying condition (1) (being a Quine atom in the AFA sense) entails satisfying condition (4) (being a Kleene fixed point), or vice versa. The two are combined by the typeclass definition — they are required together because we take them to be the same structural fact, not because one is derived from the other.

In short: "four-way equivalence" means "all four hold in any KleeneStructure instance." (1)–(3) are independently proved equivalent. (4) is bundled in by the typeclass requirement. The philosophical claim — that Kleene computational self-reference and AFA set-theoretic self-reference are the same thing — is the motivation for the typeclass design, not a consequence derived within it.

### Theorem T-COMP — Computational Grounding (ZPK.lean § III)

In any KleeneStructure lattice  $L$ , for any  $q : L$ , the following are equivalent:

- (1) IsQuineAtom  $q$  — set-theoretic self-reference (AFA)
- (2)  $q = \perp$  — order-theoretic minimum (ZP-A)
- (3)  $\forall x : L, \text{join } q \ x = x$  — algebraic generator (ZP-A A4)
- (4)  $\exists \text{ botCode} : \text{Code}, \text{IsComputationalQuine } \text{botCode}$  — computational self-reference

Note on (4): it is present in any KleeneStructure instance by typeclass requirement (botCode\_is\_quine is a required field). The equivalence of (1)–(3) is derived by T-EXEC; the presence of (4) follows from the structural commitment of KleeneStructure.

Lean: ZeroParadox.ZPK.t\_comp. Purity: [propext, Classical.choice, Quot.sound] — from Mathlib computability. ✓

## I. Why Four Languages?

DA-1's three informal paths (Path 1: AFA structural, Path 2: informational, Path 3: Kolmogorov/computational) were previously understood as three separate corroborations converging on the same conclusion. ZP-K shows they are not independent: Paths 1 and 3 are projections of one structural identity onto two different formal systems.

Path 1 says: nothing external to  $\perp$  can execute  $\perp$ , so  $\perp$  must execute itself —  $\perp = \{\perp\}$ . Path 3 says: no shorter external program generates  $\perp$  —  $\perp$  is its own minimal program. These are the same claim. " $\perp$  executes itself" (AFA language) and " $\perp$  is its own program" (computability language) are the same structural fact. The convergence of the informal paths is not coincidence — it is identity.

### Theorem: da1\_paths\_unified (ZPK.lean § IV)

In any KleeneStructure lattice:

IsQuineAtom  $\perp \wedge \text{IsComputationalQuine } \text{botCode}$

### Theorem: da1\_paths\_unified (ZPK.lean § IV)

The AFA self-containment argument (Path 1) and the Kleene computational fixed-point argument (Path 3) are the same structural fact, simultaneously witnessed by the KleeneStructure instance.

Lean: ZeroParadox.ZPK.da1\_paths\_unified. Purity: [propext, Classical.choice, Quot.sound]. ✓

## II. The Description-Instantiation Gap

The last philosophical vulnerability in the framework was the "description-instantiation gap": why does mathematical self-reference imply computational execution? The gap assumed the two were different things connected by a philosophical bridge.

They are not different things.  $\perp$  in the computational instantiation IS the universal Turing machine in its ground state. The universal Turing machine is not a description awaiting an external executor — it IS the executor. The question "why does this description execute?" is incoherent when applied to U, because U is not a description. U is the thing that executes descriptions. The question does not apply to it.

### Theorem: description\_instantiation\_gap\_closed (ZPK.lean § IV)

In any KleeneStructure lattice:

$\text{IsQuineAtom } \perp \wedge \forall q : L, \text{IsQuineAtom } q \rightarrow q = \perp$

The static-description alternative is structurally eliminated, not argued away.  $\perp$  is not a description that could await an external interpreter.  $\perp$  IS the executor — the universal Turing machine in ground state, identified structurally with the Kleene fixed point and the AFA Quine atom.

Lean: ZeroParadox.ZPK.description\_instantiation\_gap\_closed. Purity: [propext, Classical.choice, Quot.sound]. ✓

## Section IV: Axiom Footprint

All ZP-K theorems carry axioms [propext, Classical.choice, Quot.sound]. These enter exclusively through Mathlib's computability infrastructure — Kleene's theorem (`fixed_point2`) and Roger's theorem (`fixed_point`) use classical logic and the axiom of choice. They do not enter through ZPSemilattice or AFAStructure.

ZP-J T-EXEC and all its corollaries remain axiom-free. The classical axioms are entirely localised to the computational layer. The order-theoretic and set-theoretic results are unaffected.

### Remark: Classical Choice in Computability

The use of `Classical.choice` in ZP-K is structurally necessary: Kleene's theorem is an existence result, and the code witnessing the fixed point is selected non-constructively. This is standard in computability theory — the theorem guarantees existence without giving a canonical construction.

The `MachinePhase` instance (§ V) uses `Classical.choose` to pick `botCode` from `computational_quine_exists`. This makes `machinePhaseKleene` noncomputable, which is correct and expected.

## Section V: MachinePhase Instance — DA-1 Formally Closed

### I. The Concrete Instantiation

ZP-E's MachinePhase type is the two-element type {initial, running} carrying the ZPSemilattice instance (bot = initial =  $c_0$ , join = binary maximum). ZP-J gave it AFAStructure via the selfMem predicate. ZP-K gives it KleeneStructure by adding the computational witness botCode.

The selfMem definition for MachinePhase is: selfMem  $x := x = \perp$ . This is the CIC-compatible encoding of AFA self-containment: "self-containing" means "equals the bottom element." Anti-foundation is not required at the typeclass level — the relevant structural fact ( $\perp$  is the unique self-containing element) is captured by the definition and proved by rfl.

#### AFAStructure MachinePhase Instance (ZPK.lean § V)

instance machinePhaseAFA : AFAStructure MachinePhase where

selfMem  $x := x = \perp$  (self-containing = equals initial state)

quine\_unique \_\_ hx hy := hx.trans hy.symm (if  $x = \perp$  and  $y = \perp$  then  $x = y$ )

bot\_self\_mem := rfl ( $\perp = \perp$ , proved by reflexivity)

This is the CIC encoding of  $\perp = \{\perp\}$ : the initial machine state is self-containing and is the unique element with this property. No ZF+AFA axiom is added — the structural fact is encoded as a definition that Lean verifies.

#### KleeneStructure MachinePhase Instance (ZPK.lean § V)

noncomputable instance machinePhaseKleene : KleeneStructure MachinePhase where

botCode := Classical.choose computational\_quine\_exists

botCode\_is\_quine := Classical.choose\_spec computational\_quine\_exists

bot\_self\_mem\_from\_kleene := rfl

botCode is selected non-constructively from the existence proof provided by Kleene's theorem. It is the computational Quine witnessing that  $\perp$ 's self-reference has a computational expression: a program that IS its own program.

### II. DA-1 Closed

With the MachinePhase KleeneStructure instance in place, the abstract theorem da1\_computational (which holds for any KleeneStructure lattice) applies directly to ZP-E's machine. The result is concrete.

#### Theorem da1\_closed\_concrete — DA-1 Formally Closed (ZPK.lean § V)

da1\_closed\_concrete : IsQuineAtom ( $\perp$  : MachinePhase)

## Theorem da1\_closed\_concrete — DA-1 Formally Closed (ZPK.lean § V)

The initial machine state  $c_0$  is a Quine atom: it is self-containing and is the unique self-containing element of the MachinePhase lattice.

Interpretation:  $c_0$  is not a static description awaiting an external interpreter.  $c_0$  IS the executor — the universal Turing machine in its ground state, for which no external executor exists by structural definition. The description-instantiation gap is dissolved: "description awaiting execution" is not a coherent state for  $c_0$ .

Lean: ZeroParadox.ZPK.da1\_closed\_concrete. Purity: [propext, Classical.choice, Quot.sound]. ✓

### III. What Changed for DA-1

ZP-E's DA-1 section previously carried the designation "Outside Lean Scope" with three justifications: Path 1 requires ZF+AFA (incompatible with Lean's CIC/MLTT); Path 3 requires Kolmogorov complexity (uncomputable, absent from Mathlib); Path 2 requires an ontological bridge not formalizable in type theory.

ZP-K resolves Paths 1 and 3. Path 1 (AFA structural) is resolved by the AFAStructure typeclass: selfMem encodes  $\perp = \{\perp\}$  in CIC-compatible form, and the proof obligation is discharged by the MachinePhase instance. Path 3 (computational) is resolved by the KleeneStructure instance: botCode witnesses the Kleene fixed point, which is the formal expression of "no shorter program is prior to  $\perp$ ."

Path 2 (informational bridge: unbounded surprisal  $\rightarrow$  necessarily executing) is a foundational commitment — a missing principle, not a missing proof. The mathematics of L-INF (ZPC.l\_inf) is proved; but the step from "exceeds every finite informational bound" to "therefore necessarily executing" asks what it means for a mathematical structure to instantiate rather than merely satisfy conditions. No computability library answers this question. Forward paths: (a) a new axiom explicitly committing to this bridge; (b) a connection to Chalmers' notion of implementation; (c) the ZP-PQ dissolution argument — the description-instantiation gap assumes a separability that the universality of the framework dissolves. Importantly, DA-1 does not depend on Path 2: Paths 1 and 3 are formally closed, and the formal spine (DP-2 + da1\_minimal\_path) is proved axiom-free. Path 2 is motivational context; its forward resolution belongs to ZP-PQ.

DA-1 Lean scope status after ZP-K: Path 1 (structural, AFA): IN SCOPE — da1\_closed\_concrete : IsQuineAtom  $\perp$ . Path 3 (computational, Kleene): IN SCOPE — botCode\_is\_quine witnesses the fixed point. Path 2 (informational, L-INF bridge): FOUNDATIONAL COMMITMENT — a missing principle, not a missing proof. Forward: ZP-PQ dissolution argument.

### Traceability Register — ZP-K v1.2

Claim	Grounded In	Axioms	Status
selfApply_partrec	eval_part (Mathlib) + Primrec.encode + Primrec.nat_add	[propext, Classical.choice, Quot.sound]	Lean: ZPK.selfApply_partrec ✓

Claim	Grounded In	Axioms	Status
computational_quine_exists	kleene_fixed_point_exists + selfApply_partrec	[propext, Classical.choice, Quot.sound]	Lean: ZPK.computational_quine_exists ✓
T-COMP: four-way equivalence	ZP-J T-EXEC (t_exec_triple_iff)	[propext, Classical.choice, Quot.sound]	Lean: ZPK.t_comp ✓
da1_paths_unified	bot_is_quine_atom + botCode_is_quine	[propext, Classical.choice, Quot.sound]	Lean: ZPK.da1_paths_unified ✓
description_instantiation_gap_closed	bot_is_quine_atom + ZP-J t_exec	[propext, Classical.choice, Quot.sound]	Lean: ZPK.description_instantiation_gap_closed ✓
machinePhaseAFA (AFAStructure)	selfMem := x = ⊥; quine_unique; bot_self_mem := rfl	No axioms	CIC encoding of ⊥ = {⊥} for MachinePhase ✓
machinePhaseKleene (KleeneStructure)	machinePhaseAFA + Classical.choose computational_quine_exists	[propext, Classical.choice, Quot.sound]	noncomputable — Classical.choice for botCode ✓
da1_closed_concrete	da1_computational + machinePhaseKleene	[propext, Classical.choice, Quot.sound]	Lean: ZPK.da1_closed_concrete ✓ DA-1 closed

## Open Items Register — ZP-K v1.2

Item	Status	Description
DA-1 Path 1 (AFA structural)	CLOSED — da1_closed_concrete	IsQuineAtom (⊥ : MachinePhase). The initial machine state is self-containing and self-executing by structural necessity.
DA-1 Path 3 (computational)	CLOSED — machinePhaseKleene	botCode_is_quine witnesses the Kleene fixed point: no shorter program is prior to ⊥.
DA-1 Path 2 (informational)	FOUNDATIONAL COMMITMENT	L-INF (ZPC.l_inf) is proved. The bridge "unbounded surprisal → necessarily executing" is a missing principle, not a missing proof. The gap between 'system at P <sub>0</sub> ' and 'system is running' cannot be closed by any computability library. Forward paths: new axiom, Chalmers' implementation notion, or ZP-PQ dissolution argument. DA-1 does not depend on Path 2 — Paths 1 and 3 are closed.
selfApply uniqueness	CLOSED — not attempted (correct)	Computational quines are not unique in general. Uniqueness flows from ZP-J T-EXEC (set-theoretic side). No uniqueness theorem for computational quines is needed or appropriate.
Roger's fixed-point theorem	CLOSED — roger_fixed_point_exists	For any computable $f : \text{Code} \rightarrow \text{Code}$ , $\exists c$ , $\text{eval}(f\ c) = \text{eval}\ c$ . Lean: ZPK.roger_fixed_point_exists — [propext, Classical.choice, Quot.sound]. ✓

Item	Status	Description
ZP-B MachinePhase instance	OPEN — future work	The 2-adic model from ZP-B ( $\mathbb{Q}_2$ structure) has not been given a KleeneStructure instance. This is a natural extension but not required for T-SNAP or DA-1.

---

End of ZP-K v1.2 | Computational Grounding of Self-Reference | DA-1 closed: `da1_closed_concrete : IsQuineAtom ( $\perp$  : MachinePhase)` | Four-way equivalence: Quine atom =  $\perp$  = join identity = Kleene fixed point | Path 2 recharacterized: foundational commitment, not missing proof; forward: ZP-PQ | All ZPK.lean theorems verified. Axioms: [propext, Classical.choice, Quot.sound]