
A CYCLE IS NOT A SOURCE

The Least-Provenance Theorem for Cyclic Records and the Boundary of Self-Reference

Kirill Shebetov · beforeword / research

Phase 3 · English reading edition · 30 August 2026

Contents

Abstract

0. No Exemption for Words
1. The Exact Claim Being Refuted
2. The Progression from Phase 0 to Phase 3
3. Finite Typed Graph with Cycles
4. Transfer Operator and Least Fixed Point
5. CP-LFP-3 (CFP-3): Least-Provenance Theorem
6. Strongly Connected Components
7. An Unseeded Self-Loop
8. An Unseeded Mutual Cycle
9. A Fixed Point Is Not a Selected Fixed Point
10. SG-LFP-3 (GST-3): Grounded Support
11. Cyclic Provenance Cut
12. Composition with AST-1
13. Self-Reference Receives No Exemption
14. Mathematical Reading as an Input
15. Full Concession to Technical Success
16. The Question “What For?” (Exact Russian Coordinate: зачем)
17. Iteration, Duration, and the Limit
18. Strongest Objections
19. Exact Falsifiers
20. Executable Domain
21. Self-Application
22. Predecessors and the Novelty-Status Record
23. Boundary of the Claims
24. Conclusion

Source Status

CP-LFP-3 (CFP-3): Least-Provenance Theorem for a Cyclic Record

0. Status of the notation
1. Relation to Phases 0–2
2. Finite cyclic record
3. Cyclic-provenance operator
4. CP-LFP-3: path theorem
5. A cycle is not a source
6. SCC condensation
7. A cut in a cyclic graph
8. Composition with AST-1
9. Complete closure does not create an external target

10. Covariance under renaming
11. Snapshots and the exact boundary of non-retroactivity
12. Admission gates
13. Normative statuses
14. Required controls
15. Exact falsifiers
16. Self-application and the maximal line

SG-LFP-3 (GST-3): Support and Protected-Introduction Theorem

0. Normative status
1. Why an AND/OR layer is needed
2. Finite support contract
3. Least AND/OR closure
4. Finite support tree
5. SG-LFP-3: support-tree theorem
6. Minimal root bases
7. Exact blocking of the root
8. Cycles in an AND/OR hypergraph
9. CP-LFP-3 as a unary special case
10. Binding, event, cause, application, purpose, and meaning
11. Self-reference
12. Admission gates
13. Normative statuses
14. Required controls
15. Exact falsifiers
16. Maximal formulation

FP-SEL-3 (FPS-3): fixed-point selection and circular residue

0. Normative status
1. Base positive contract
2. The interval of fixed points
3. Support-grounded fixed point
4. Unsupported circular residue
5. Minimal examples
6. Fixed-point selection as input
7. Target selection after the fixed point
8. Self-reference and quotation
9. Feedback, controller, and rocket
10. Admission gates
11. Normative statuses
12. Required controls
13. Exact falsifiers

14. Maximal formulation

SELF-PROOF IS NOT A SOURCE

Self-Reference, Mutual Checking, Mathematical Reading, the Rocket, and the Question “What For?” (Exact Russian Coordinate: зачем)

Abstract

0. The lines are supplied first

1. Relations that must not be conflated

2. The exact Phase 0 → Phase 1 → Phase 2 → Phase 3 line

3. A single node declaring itself proved

4. A self-loop with an explicit seed

5. Two nodes checking one another

6. A ring of multiple checkers

7. Support graph: a reference is not a ground

8. A proof checker checking its own code

9. Self-proof and levels of claim

10. Fixed-point policy as part of the proof contract

11. Mathematics begins in this record as a reading contract

12. Internal semantics is admitted in full

13. The strongest case for the rocket argument

14. The rocket feedback loop

15. What exactly follows from a successful flight

16. The Question “What For?” After the Full Concession (Exact Russian Coordinate: зачем)

17. One hundred, two hundred, three hundred passes

18. Constitutive convention

19. Coinduction

20. Mutual verification and independence

21. Self-modification and a changing graph

22. The strongest objections

23. Exact application falsifiers

24. Self-application of this file

25. Boundary of the application claims

26. Conclusion

Source status

05. ADVERSARIAL REVIEW AND SELF-APPLICATION OF PHASE 3

A CYCLE IS NOT A SOURCE

0. Purpose and fixed boundary

1. Exact objects under attack

2. Red team against CFP-3

3. Red team against GST-3

4. Red team against fixed-point selection

5. Hidden readset, an unclosed chain, and regress
6. Payload spoof and substitution of CP-sort / SG-role
7. Maximum concession: the rocket and successful application
8. Performative contradiction and self-application
9. Failure conditions
10. Mandatory positive and negative controls
11. Exact claim / status / falsifier register
12. Final red-team boundary

Abstract

Phase 2 localized a distinguishing typed atom among the ancestors of a conclusion, but only for a finite acyclic graph. The present study removes exactly that restriction. Finite directed graphs may contain self-loops, mutual references, and strongly connected components. The words цикл, источник, истина, доказательство, причина, цель, and смысл—read as “cycle,” “source,” “truth,” “proof,” “cause,” “goal,” and “meaning”—are not taken for what they name. They enter only as exact records and technical codes in the declared contract.

Each node has protected origin-tagged introductions. A monotone transfer operator is defined on the complete lattice of provenance states: a node contains its own introductions and everything received along recorded incoming edges. Provenance is selected not as an arbitrary fixed point, but as the least fixed point obtained by iteration from the empty state.

CP-LFP-3 (CFP-3) records the cyclic-provenance path theorem. An origin-tagged atom belongs to the least provenance of a node iff the atom is introduced in a protected field at some source node and a finite directed path reaches the node under consideration from that source. A cycle may carry an introduced atom repeatedly, but a source-free strongly connected component does not generate it by circulation alone.

For finite positive AND/OR hypergraphs, after normalization of nullary constructors and with nonempty premises for ordinary rules, **SG-LFP-3 (GST-3)** records that every conclusion in the least closure has a finite support tree whose leaves are declared seeds. Pure self-support or mutual support without a protected seed and without an alternative finite seed-grounded support tree does not enter μT_S . Alternative minimum bases are retained as exact sets of recorded grounds.

Finally, **FP-SEL-3 (FPS-3)** separates solving a fixed-point equation from selecting a fixed point. For support operator T_S , $X \in \text{Fix}(T_S)$ is support-grounded iff $X = \mu T_S$. If several fixed points are admissible, the rule least, greatest, stable, or any other selection rule is an added input. When $X \neq \mu T_S$, every code in the nonempty residue $X \setminus \mu T_S$ lacks a finite seed-grounded support tree.

The result is applied to self-proof, mutual checking, recursive definitions, a mathematical reading contract, technical application, and the question зачем (“what for?”). It does not declare cycles useless or deny the conditional force of recursive mathematics. It registers a result for the following declared claim: a closed record does not become the first source of a missing protected atom by itself.

Keywords: record boundary; cyclic provenance; least fixed point; strongly connected component; origin tag; grounded support; self-reference; mutual checking; fixed-point policy; application binding; goal; self-application.

0. No Exemption for Words

The strings are presented first:

цикл
самоссылка
истина
доказательство
проверено
ракета
событие
цель
смысл

They are read as “cycle,” “self-reference,” “truth,” “proof,” “checked,” “rocket,” “event,” “goal,” and “meaning.” No string presents what it names. The string `цикл` (“cycle”) does not form a cycle; a cycle appears only after a node set and exact edges have been declared. The string `истина` (“truth”) receives no truth status. The string `доказательство` (“proof”) is not a proof object. The string `ракета` (“rocket”) remains a string even if, outside this file, the full concession that a rocket exists and flies is accepted.

The same restriction applies to the formal words in this study. `Graph`, `atom`, `origin`, `fixed point`, `least`, `support`, and `source` are role labels. Writing them does not establish correctness of a model, correspondence outside the record, or priority of the study.

At level zero an exact carrier is fixed. A separate reading contract then introduces:

- nodes;
- directed edges;
- atom types;
- protected introduction fields;
- a transfer rule;
- a fixed-point policy;
- a criterion for accepting a result.

Before that contract there is a carrier. After it there is a record read under the declared contract. Reverse transfer is prohibited: the later label `источник` (“source”) does not turn an earlier payload into a source.

1. The Exact Claim Being Refuted

Phase 3 does not use the unbounded line `любой цикл ничего не создаёт` (“no cycle creates anything”). That line is false even inside the record contract: a cycle node may have a local introduction, a cycle may

receive an atom along an incoming edge, and a recursive procedure may correctly compute a new payload from introduced data.

The exact claim refuted is **cyclic self-origin**:

A protected atom enters the least provenance of a node solely because records refer to one another cyclically, even though no reaching node has a protected introduction of that atom.

Under least monotone semantics, the negative result is exact. If an atom is present, there is an origin-tagged introduction and a finite path from its origin to the node where it is present. Without such an introduction, circulation does not fill empty provenance.

The word полностью (“fully”) below applies to the universal quantifier over finite directed graphs and atoms satisfying the declared positive-union contract. It does not apply to an unrepresented class of every possible use of the words цикл (“cycle”), математика (“mathematics”), or истина (“truth”).

2. The Progression from Phase 0 to Phase 3

The four phases do not repeat one thesis. Each addresses a specific limitation of the preceding phase.

Phase 0: Bridge Provenance

MRB-0 separated formal type M , application type A , and target type V . The only introduction rule for A required a bridge, and dependencies were combined by union without deletion. An accepted proof object of type A therefore could not have an empty set of bridge dependencies.

Residue: the result depended on a particular type architecture and required a separate transport τ to another corpus.

Phase 1: Functional Boundary

RBI-1 replaced the architectural prohibition with a general factorization theorem. For complete available input p and independently specified target Q ,

$$Q = g \circ p \Leftrightarrow \ker p \subseteq \ker Q.$$

One pair with identical complete input and different target excludes every evaluator of that input. A longer calculation, internal model, oracle answer, randomization, or transcript helps only when it brings in a distinguishing channel.

Residue: the added difference was required, but its place of introduction was not localized.

Phase 2: Ancestral Localization

AST-1 introduced origin-tagged ancestry in a finite immutable DAG. If a correct conclusion has the form

$$Q = C_v = F(\rho, \eta_v),$$

then

$$\ker \rho \cap \ker \eta_v \subseteq \ker Q.$$

A distinguishing typed atom therefore has an origin among the reflexive ancestors of the conclusion. FDT-2 separately localized the first selectable stage of a finite filtration.

Residue: the DAG excluded self-loops and mutual references. The ordinary word предок (“ancestor”) ceased to distinguish nodes inside a strongly connected component. Phase 2 explicitly left cycles to a separate domain requiring fixed-point semantics.

Phase 3: Provenance Inside Cycles

Phase 3 retains origin tags while replacing recursive ancestor union with the least fixed point of a monotone operator. A strongly connected component can circulate an atom, but its origin remains the node of protected introduction. If several fixed points are admissible, selection is brought out of hiding and entered in the complete readset.

The resulting line is:

```
типовой bridge
-> различающий channel
-> origin в DAG
-> origin и policy в циклическом graph
```

Occurrence-local English reading:

typed bridge -> distinguishing channel -> origin in a DAG -> origin and policy in a cyclic graph

This English reading is a separate written surface and does not replace the exact fenced source sequence.

This expands the declared domain. It does not expand words into what they name.

3. Finite Typed Graph with Cycles

Let

$$G = (V, E)$$

be a finite directed graph. An edge

$$u \rightarrow v$$

means only a recorded dependency: node v declares u as an input to transfer. It does not mean physical causation, temporal precedence, trust, or truth.

The graph may contain:

- self-loops $v \rightarrow v$;
- mutual edges $u \rightarrow v \rightarrow u$;
- arbitrary finite strongly connected components;
- acyclic regions between components.

The notation T_3 denotes exactly the eight Phase 3 sorts:

- *FORMAL, BINDING, EVENT, GOAL*;
- *REFERENT, CAUSE, APPLICATION, MEANING*.

In formulas these are uppercase mathematical sort codes. The JSON field `sort` serializes them under the fixed bijection $\text{FORMAL} \leftrightarrow \text{"formal"}$, $\text{BINDING} \leftrightarrow \text{"binding"}$, $\text{EVENT} \leftrightarrow \text{"event"}$, $\text{GOAL} \leftrightarrow \text{"goal"}$, $\text{REFERENT} \leftrightarrow \text{"referent"}$, $\text{CAUSE} \leftrightarrow \text{"cause"}$, $\text{APPLICATION} \leftrightarrow \text{"application"}$, $\text{MEANING} \leftrightarrow \text{"meaning"}$. The top-level node field `kind` uses separate uppercase literals and is not the `sort` field.

Sort names do not establish their corresponding relations. *EVENT* is the sort of an event code, not an event. *REFERENT* is the sort of a referent code; *BINDING* remains a separate sort. *MEANING* is the sort of a meaning code, not meaning.

For every $\tau \in T_3$, fix a finite set of permitted atom ids A_τ . The protected local introduction at node v is

$$\text{Intro}_\tau(v) \subseteq A_\tau.$$

Payload is inert. Writing `goal`, `true`, `proved`, `rocket`, or a complete atom serialization in payload creates no introduction. A protected introduction field and payload are different coordinates of the record.

The origin-tagged universe is

$$\Omega = V \times \bigcup_{\tau \in T_3} (\{\tau\} \times A_\tau).$$

The local introduction set at node v is

$$\text{OIntro}(v) = \{(v, \tau, a) : \tau \in T_3, a \in \text{Intro}_\tau(v)\}.$$

The origin tag is retained during transfer. The same atom code introduced at two nodes yields two different origin-tagged records.

4. Transfer Operator and Least Fixed Point

A provenance state is a family

$$X = (X_v)_{v \in V} \in L = (2^{\mathcal{Q}})^V.$$

The order on L is coordinatewise inclusion. Its bottom element is

$$\perp_v = \emptyset.$$

The transfer operator is

$$\Phi_G(X)_v = OIntro(v) \cup \bigcup_{u \rightarrow v} X_u.$$

It is monotone:

$$X \subseteq Y \Rightarrow \Phi_G(X) \subseteq \Phi_G(Y).$$

Iteration begins not from a desired result but from the empty state:

$$X^{(0)} = \perp, X^{(n+1)} = \Phi_G(X^{(n)}).$$

Because V and Ω are finite in the declared artifact contract, the ascending sequence stabilizes after finitely many strict additions. Define

$$P^\mu = \mu \Phi_G = \bigcup_{n \geq 0} X^{(n)}.$$

The word *least* is not an evaluative adjective here. It means the inclusion-minimal fixed point of Φ_G . This policy is recorded before the result is computed.

5. CP-LFP-3 (CFP-3): Least-Provenance Theorem

For every $s, v \in V$, $\tau \in T_3$, and $a \in A_\tau$,

$$(s, \tau, a) \in P_v^\mu \Leftrightarrow a \in Intro_\tau(s) \wedge s \rightsquigarrow v.$$

Here $s \rightsquigarrow v$ is a finite directed path, including a path of length zero when $s = v$.

Derivation Outline

After the first iteration, only local introductions are present. After the second, introductions that have traveled along paths of at most one edge are present. Induction on n gives

$$(s, \tau, a) \in X_v^{(n+1)}$$

exactly when a was introduced at s and a path $s \rightsquigarrow v$ of length at most n exists. Taking the union of all finite iterations gives one direction. The other follows by traversing any finite path: the introduction enters at its source and is transferred along each edge.

The complete derivation, monotonicity lemmas, and contract conditions are placed in the separate Phase 3 theorem layer. The central consequence recorded here is:

Membership of an atom in P_v^μ has two mandatory fields: an introduction and a finite transfer path. A cycle replaces neither.

6. Strongly Connected Components

Let $SCC(v)$ be the strongly connected component of node v . Contracting every SCC to one vertex yields the condensation graph

$$Cond(G),$$

which is a DAG.

CP-LFP-3 gives an exact account of component provenance:

1. an atom may be introduced locally inside an SCC;
2. an atom may enter along a path from an upstream component;
3. after entry, it may propagate to reachable nodes of the SCC;
4. absence of both local and upstream introduction leaves it absent throughout the SCC.

The phrase *в цикле нет первого узла* (“there is no first node in a cycle”) therefore creates no exception. A chronologically first node inside the closed record may be absent from the model, but an atom’s origin is not *первый по кругу* (“first around the loop”). It is the node of protected introduction. Its tag survives any number of traversals.

Write $s \rightsquigarrow C$ iff $s \rightsquigarrow v$ for some $v \in C$. The exact SCC consequence is

$$(\nexists s \in V : a \in Intro_\tau(s) \wedge s \rightsquigarrow C) \Rightarrow \forall v \in C \nexists s \in V : (s, \tau, a) \in P_v^\mu.$$

A source-free component may be structurally complex, contain thousands of edges, and admit infinitely many described traversals. Route complexity is not an introduction.

7. An Unseeded Self-Loop

Consider one node v and an edge $v \rightarrow v$. Suppose

$$OIntro(v) = \emptyset.$$

Bottom-up iteration gives

$$X_v^{(0)} = \emptyset, X_v^{(n+1)} = X_v^{(n)} = \emptyset,$$

and

$$P_v^\mu = \emptyset.$$

The record may contain payload:

```
this node proves this node
TRUE
VALID
MEANING
```

Payload does not change the result. Atom `VALID` enters P_v^μ only through a local or upstream introduction. Another fixed-point policy may select another state, but it does not change $\mu\Phi_G$ for the existing instance.

This does not mean that self-referential formal objects are prohibited. It records only the absence of an origin under this least-provenance contract.

8. An Unseeded Mutual Cycle

Let

$$u \rightarrow v, v \rightarrow u,$$

and let both local introduction sets be empty. The payload may contain the exact strings

```
u: v проверен
v: u проверен
```

These strings are read as “u: v is checked” and “v: u is checked.” Least iteration still begins with the empty state and remains empty.

If u instead introduces atom a in a protected field, then

$$(u, \tau, a) \in P_u^\mu \text{ и } (u, \tau, a) \in P_v^\mu.$$

The Russian conjunction embedded in the formula means “and.”

The cycle now transfers a protected introduced atom. It does not hide its origin: tag u remains at both nodes.

This gives the polarity symmetry:

- a cycle without a local/upstream introduction receives no atom;
- a cycle with a local/upstream introduction legitimately propagates the atom;
- presence of an introduction does not turn an atom code into a referent, event, or truth.

9. A Fixed Point Is Not a Selected Fixed Point

The set of fixed points is

$$\text{Fix}(\Phi_G) = \{ X : \Phi_G(X) = X \}.$$

The equation may have more than one solution. For an unseeded self-loop it becomes

$$X_v = X_v.$$

The empty state satisfies this record; if $\Omega \neq \emptyset$, nonempty states may satisfy it as well. The equality itself selects none of them.

A contract that selects one fixed point as its accepted state must supply the field

```
fixed_point_policy
```

For example:

```
least
greatest
stable
declared-selector:<id>
```

These strings are policy codes. Writing one does not establish correctness of the policy, but omitting the policy leaves the selection contract's readset incomplete. In CP-LFP-3 the policy is fixed in advance as least-from-bottom.

Unary Path-Grounded Consequence and FP-SEL-3

For a positive-union operator of the form Φ_G , the only fixed point in which every membership has a finite introduction-to-node path witness is $\mu \Phi_G$. This is the unary path-grounded consequence of CP-LFP-3.

Thus, if

$$X \in \text{Fix}(\Phi_G), X \neq P^\mu,$$

there is a circular residue

$$X \setminus P^\mu$$

without a finite-path introduction witness. In a finite graph, tracing dependencies backward reaches a cycle inside the unsupported residue.

Normative **FP-SEL-3** concerns support operator T_S , defined next:

$$X \in \text{Fix}(T_S) \wedge X \text{ support-grounded} \Leftrightarrow X = \mu T_S.$$

If $X \in \text{Fix}(T_S)$ and $X \neq \mu T_S$, every $q \in X \setminus \mu T_S$ lacks a finite seed-grounded support tree. Other seeds do not alter this. For example, a rule $\{q, s\} \Rightarrow q$ with $s \in S$ retains the mandatory circular branch for q .

This does not make a nonleast fixed point ложной ("false"). It may be accepted under a coinductive or other policy. The exact result is:

A policy admitting unsupported residue is an added input; closure of the equations alone did not introduce the residue.

10. SG-LFP-3 (GST-3): Grounded Support

An ordinary directed graph transfers individual atoms. Rules with several premises use a finite monotone AND/OR hypergraph. The abstract theorem fixes a finite set of code ids N , a seed set $S \subseteq N$, and a finite rule set R . For every $r \in R$,

$$\emptyset \neq \text{Prem}(r) \subseteq N, \text{Head}(r) \in N.$$

The operator is

$$T_S(X) = S \cup \{ \text{Head}(r) : r \in R, \text{Prem}(r) \subseteq X \}.$$

In the executable refinement, the base and claim namespaces are separated by the tagged embedding

$$N = B \sqcup C, S = B.$$

Here B contains tagged `base_id` values and C tagged `claim_id` values. For an inference, $\text{Prem}(r)$ is the union of `required_base_ids` in B and `premise_claim_ids` in C , while `conclusion_id` gives $\text{Head}(r) \in C$. The executable checker therefore reports the claim projection of the abstract least closure without erasing the BASE/CLAIM distinction during checking.

The condition $\text{Prem}(r) \neq \emptyset$ is mandatory. An ordinary empty-premise rule lies outside the SG contract. Before serialization, a nullary constructor is normalized into a separate protected base such as `RULE-SEED:<rule-id>`, and the inference to its conclusion explicitly lists that `base_id` in `required_base_ids`. If `premise_claim_ids` and `required_base_ids` are both empty, the checker does not perform normalization for the author. It emits `INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT` with reason code `NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE`.

SG-LFP-3 records:

Every root code in the least closure has a finite support tree; internal vertices correspond to declared rules, and every leaf is a declared seed.

The claim self-loop

$$C : q \Rightarrow C : q$$

without a rule path from S gives $C : q \notin \mu T_S$. The mutual claim pair

$$C:p \Rightarrow C:q, C:q \Rightarrow C:p$$

without a rule path from S gives $C:p, C:q \notin \mu T_S$. A positive executable-compatible control uses a separate base:

$$B:b \in S, B:b \Rightarrow C:p, C:p \Rightarrow C:q, C:q \Rightarrow C:p.$$

Both claim codes then enter least closure, and every presented support tree ends at leaf $B:b$, not claim code $C:p$.

If a root has several alternative minimum seed bases, they form an inclusion antichain. Selecting one convenient basis does not erase the others or make the root unconditional.

SG-LFP-3 concerns support records. The word *premise* does not present a fact; the word *proof* does not present the truth it names.

11. Cyclic Provenance Cut

Let $Prov_H^\mu(x) = \{(\tau, a) : \exists s(s, \tau, a) \in P_H^\mu(x)\}$ be the untagged projection. Fix root r , untagged code (τ, a) , and its root-reaching introducers

$$S_{\tau,a}^r = \{s \in V : a \in Intro_\tau(s) \wedge s \rightsquigarrow_G^* r\}.$$

Let $P_G(s, r)$ be the finite directed paths from s to r , and $V(p)$ the vertex set of path p .

Under the internal-cut contract, every node in $S_{\tau,a}^r$ and the root is protected. The permitted deletion is

$$B \subseteq V \setminus (S_{\tau,a}^r \cup \{r\}),$$

without rewiring, followed by recomputation of least provenance on $G - B$. Then

$$(\tau, a) \notin Prov_{G-B}^\mu(r) \Leftrightarrow \forall s \in S_{\tau,a}^r \forall p \in P_G(s, r), V(p) \cap B \neq \emptyset.$$

If $S_{\tau,a}^r = \emptyset$, cut status is **ABSENT** and rank is **null**: an empty hitting set is not presented as a cut certificate. If an introducer is the root or has a direct edge to it, no admissible internal cut exists; the status is **UNBLOCKABLE-PROTECTED-ENDPOINT**, not cut rank zero. In all other cases, the exact minimum receives **CUT-CERTIFIED**. Cycles do not alter the criterion: every root-reaching walk contains a simple path. A minimum combined internal cut measures only structural redundancy among recorded routes, not truth, importance, physical causation, or meaning.

12. Composition with AST-1

Let $K \neq \emptyset$ be the declared class of exact-record indices. For conclusion v , define the cyclic ancestry projection

$$\eta_v^\mu(k) = \text{canon}(P_v^\mu(k)).$$

Suppose ρ contains the evaluator's complete readset, F is total, and for every $k \in K$,

$$C_v(k) = F(\rho(k), \eta_v^\mu(k)) = Q(k).$$

The Phase 1 functional theorem is unchanged:

$$\ker \rho \cap \ker \eta_v^\mu \subseteq \ker Q.$$

If $\rho(k_0) = \rho(k_1)$ but $Q(k_0) \neq Q(k_1)$, then $\eta_v^\mu(k_0) \neq \eta_v^\mu(k_1)$. By CP-LFP-3, a distinguishing origin-tagged member of this projection has a protected introduction and a path to the conclusion.

If another fixed-point policy is used, its identifier, order, admissibility rules, and selector belong in ρ . A hidden replacement of *least* by *greatest* is a change of input, not a refutation of the theorem.

13. Self-Reference Receives No Exemption

The string

эта строка доказана этой строкой

This exact string is read as “this string is proved by this string” and may be a graph node. A self-loop may be recorded. A parser may accept its form. A checker may classify it under a previously specified recursive contract.

No protected atom *PROVED* follows if:

- it is not introduced locally;
- it is not received along a path from another introduction;
- least closure has no seed-support tree;
- the policy contains no separate rule instituting such a status.

If an instituting rule is recorded, it is a new input. The status may then be correct **within that contract**.

Phase 3 does not prohibit instituted statuses; it fixes their dependency.

The mutual phrase *А подтверждает В, В подтверждает А* (“A confirms B, B confirms A”) likewise does not form an independent ground. With a trusted seed, kernel rule, certificate root, or external-attestation record,

the cycle may carry an introduced status. Its origin remains in a protected introduction or seed; policy separately remains in the readset and may select a nonleast state.

The detailed application appears in 04_APPLICATION_SELF_PROOF_RU.md.

14. Mathematical Reading as an Input

The structural order remains

exact carrier < reading contract < mathematical syntax
mathematical syntax < formal rules < fixed-point policy < checked status .

The word `математика` (“mathematics”) does not appear at level zero. A carrier receives status `READ-AS-MATHEMATICAL` only after parser, syntax, and rule records are supplied. If these records refer cyclically to their results, CP-LFP-3 records the provenance of protected atoms, while FP-SEL-3 records the role of policy.

A formal system may correctly derive a theorem about its own record, checker, or semantics. The result is retained as

FORMAL - STATUS - IN - DECLARED - CONTRACT

and is not renamed as

REFERENT - PRESENT
EVENT - OCCURRED
CAUSE - ESTABLISHED
GOAL - JUSTIFIED
MEANING - PRESENT

without separate relations and inputs.

The phrase `математика возникла из языка` (“mathematics arose from language”) is not used as a historical premise. Only an access order inside the supplied record is recorded: mathematical reading requires a reading contract and therefore cannot ground its own appearance in that same contract.

15. Full Concession to Technical Success

For the application, suppose all of the following are granted:

- the rocket exists;
- the rocket was built;
- the formal calculation is correct under its contract;
- the calculation was used in design and control;
- the controller, telemetry, and checking chain worked;
- the trajectory matched the calculation under the accepted criterion;

- the rocket is flying.

These items enter as a full concession, not as consequences of strings in this manuscript.

Even under this concession, the records remain distinct:

$$m = \text{formal}, b = \text{binding}, e = \text{event}, c = \text{causal-status}, g = \text{goal}.$$

A cyclic chain simulation $\leftrightarrow$ controller $\leftrightarrow$ telemetry $\leftrightarrow$ model may be technically useful and mutually dependent. Here event, fit, and application are atom/claim-role codes, each separately assigned one of the eight sorts in T_3 ; they introduce no implicit sorts. The corresponding protected atoms enter through sensor, binding, criterion, and protocol records. Circulation propagates them; it does not place them retroactively inside an early formula record.

Success is not rejected. Its reverse transfer into the earlier carrier is.

16. The Question “What For?” (Exact Russian Coordinate: *зачем*)

Cyclic repetition of a goal string is not its justification. The following remain separate:

GOAL - CODE - RECORDED
ENDORSEMENT - CODE - RECORDED
FIT - CODE - CHECKED
JUSTIFICATION - CODE - ACCEPTED
MEANING - CODE - RECORDED

These labels are claim-role codes, not additional sorts beyond T_3 . Every status receives a separately declared sort, introduction rule, and support bases. A goal may be a protected seed for optimization. Optimization may correctly compute a trajectory relative to that goal. It does not select why the goal should be accepted.

If the only rules with heads $C : g$ and $C : d$ form the mutual pair

$$C : g \Rightarrow C : d, C : d \Rightarrow C : g,$$

and neither claim has a rule path from S , then

$$C : g, C : d \notin \mu T_S.$$

Unrelated bases may belong to S , and other claims may be derived from them; the statement excludes only $C : g$ and $C : d$ under the specified absence of a support path. If a policy institutes coinductive acceptance, that policy is a separate input. If a goal enters through a protected base and separate inference, its exact supporting base/base_id is presented explicitly; merely listing the goal as a claim premise does not introduce it.

The full rocket concession therefore does not close *зачем* (“what for?”) through one technical loop. It permits a successful causal-technical chain to be recorded under an accepted contract; goal, endorsement, and justification remain separate coordinates.

17. Iteration, Duration, and the Limit

On a finite graph with finite Ω , iteration of Φ_G from $\perp$ stabilizes after finitely many strict additions, no more than the number of possible node/atom memberships. After stabilization, one more, a hundred more, or a million more traversals add no absent atom.

The strings

```
100 лет
200 лет
300 лет
бесконечно
```

These strings are read as “100 years,” “200 years,” “300 years,” and “infinitely.” They are not introductions. If the graph, local introductions, and policy remain unchanged, repeating the same operator retains the same least fixed point.

No statement about every undescribed future follows. A new sensor, goal, rule, node, oracle answer, or policy forms a new input and a new contract instance; a new graph arises precisely when its V or E changes. Phase 3 records an invariant of the declared contract, not a prohibition on any later record.

Nor does the finite checker cover an arbitrary infinite or non- ω -continuous semantics. A limit rule, transfinite stage, or infinite read channel must be declared separately; its name is not its implementation.

18. Strongest Objections

18.1. “Recursion Really Does Create New Results” (Exact Russian Source Wording: Рекурсия действительно создаёт новые результаты)

Yes. New payloads and derived codes may arise from seeds under declared rules. CP-LFP-3 concerns atoms admitted through *Intro* and retains their origin/path; multi-premise derivation and its seed tree belong to SG-LFP-3.

18.2. “The Cycle Contains a Local Introduction” (Exact Russian Source Wording: В цикле есть local introduction)

Then the SCC is not source-free. The atom legitimately enters least provenance and propagates. This is a positive control, not a refutation.

18.3. “A Greatest Fixed Point Contains an Atom Without a Seed” (Exact Russian Source

Wording: Greatest fixed point содержит atom без seed)

This may occur under a coinductive policy. For $X \in \text{Fix}(\Phi_G) \setminus \{\mu \Phi_G\}$, every origin-tagged member of $X \setminus \mu \Phi_G$ lacks a finite introduction-path witness. For $X \in \text{Fix}(T_S) \setminus \{\mu T_S\}$, every code in $X \setminus \mu T_S$ lacks a finite seed-support tree. These are different statements for unary provenance and a support hypergraph. If a nonleast state is accepted, the selection policy enters the complete input. The error would be to call it silently a result of CP-LFP-3 or SG-LFP-3.

18.4. “Fixed-Point Semantics Is Standard and Is Not a Channel” (Exact Russian Source

Wording: Fixed-point semantics стандартна и не является каналом)

Standardness is another classification string. Two contracts with identical equations and different policies may select different states. Policy therefore distinguishes outputs and belongs in the readset regardless of prevalence.

18.5. “The Self-Checking Checker Checked Itself” (Exact Russian Source Wording:

Самопроверяющийся checker проверил себя)

An exact self-check trace may be accepted and assigned a status under the checker contract. Mutual or recursive structure of the trace does not turn that status into independent evidence about compiler, runtime, hardware, referent, or world. Relevant records may be added, in which case they are inputs.

18.6. “The Cyclic System Acts on the Rocket” (Exact Russian Source Wording: Циклическая система действует на ракету)

Action and success are accepted in the full concession. They concern the expanded chain of formal, binding, implementation, sensor, event, and criterion records. An early formula code does not become the whole chain.

18.7. “A Constitutive Convention Really Does Create a Status” (Exact Russian Source Wording:

Constitutive convention действительно создаёт status)

Yes. If a convention institutes a particular loop as a victory, proof code, or admissible recursive object, formal status may be determined inside that practice. The convention is a rule/policy input; an introduction arises only through a separate protected Intro constructor. The name of the convention does not present an independent referent.

18.8. “There Is No First Ancestor in a Cycle” (Exact Russian Source Wording: В цикле невозможно найти первого предка)

CP-LFP-3 requires no first node in a cyclic order. It requires a source introduction and finite path. SCC condensation localizes an upstream or local source without artificially cutting the cycle.

18.9. “Negation or Default Rules Change the Result” (Exact Russian Source Wording: Negation или default rules меняют результат)

Correct. CP-LFP-3 concerns the declared monotone positive-union operator. Negation, defaults, defeat, time-dependent rules, or nonmonotone updates require a separate semantics and inclusion of policy in ρ . Applying the theorem without this premise is prohibited.

19. Exact Falsifiers

19.1. Countermodels to the Theorems

CP-PATH-FALSIFIER is a finite exact object satisfying the CP admission gates and using the actual $P^\mu = \mu \Phi_G$, in which at least one condition holds:

1. $(s, \tau, a) \in P^\mu_v$, but $a \notin \text{Intro}_\tau(s)$ or there is no finite path $s \rightsquigarrow_G^* v$;
2. $a \in \text{Intro}_\tau(s)$ and $s \rightsquigarrow_G^* v$, but $(s, \tau, a) \notin P^\mu_v$.

SG-SUPPORT-FALSIFIER is finite N, S, R satisfying SG-domain and NONNULL for which some $q \in N$ has at least one of:

1. $q \in \mu T_S$, but no finite support tree with leaves in S exists;
2. a valid finite support tree for q with leaves in S is presented, but $q \notin \mu T_S$.

FP-GROUNDING-FALSIFIER is a fixed point $X \in \text{Fix}(T_S)$ that is support-grounded while $X \neq \mu T_S$, or $X = \mu T_S$ while not support-grounded. The second branch also supplies SG-SUPPORT-FALSIFIER.

These are mathematical countermodels. Their definition does not depend on any checker accepting its own output.

19.2. Executable-Conformance Defects

The following refute implementation conformance to the published contract but are not, by themselves, theorem countermodels:

- a checker accepts a declared_fixed_point different from the untagged projection of computed $\mu \Phi_G$;
- a checker extracts a protected atom from payload or accepts a prohibited kind/sort;
- a checker accepts a nonmonotone operator under CP-LFP-3;
- the support checker accepts an inference with both premise_claim_ids and required_base_ids empty instead of status INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT and reason code NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE;
- the support checker accepts an incorrect claim projection of least closure, an incorrect support witness, or an incomplete antichain of minimum bases.

For an implementation defect to become a theorem falsifier, a mathematical object retaining every theorem premise and violating the corresponding equivalence must be supplied separately.

The following are not falsifiers: adding a local seed; adding an upstream introduction; changing the graph; selecting a greatest/coinductive policy instead of least; adding a hidden oracle; changing payload together with a protected field; or supplying a narrative about a referent, event, or cause outside the declared relation. Each changes a premise or input.

20. Executable Domain

Phase 3 publishes two main executable layers.

Cyclic Provenance

`cyclic_provenance_checker.py` checks:

- finiteness and exact graph structure;
- protected introductions and payload inertness;
- bottom-up iteration of the positive-union operator;
- equality of every `declared_fixed_point` to the untagged projection of the least fixed point;
- SCC condensation;
- one deterministic shortest-lex simple-path witness for every reaching introducer and the exact minimum combined internal vertex cut with protected sources/root;
- separation of nonleast residue.

Principal status strings:

```
VALID-CYCLIC-PROVENANCE-LFP
INVALID-CYCLIC-PROVENANCE
INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT
CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS
```

Controls cover unseeded/seeded self-cycle, unseeded/seeded mutual cycle, nonleast residue, payload spoof, and an acyclic baseline.

Support Grounding

`support_grounding_checker.py` checks a finite monotone inference hypergraph; seed set and least closure; support trees; minimum root bases; alternative and dominated bases; unseeded/seeded cycles; application/rocket support with and without a goal; and payload spoof.

Principal status strings:

```
VALID-SUPPORT-GROUNDING-LFP
INVALID-SUPPORT-GROUNDING
INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT
SUPPORT-GROUNDING-INVARIANT-AUDIT-PASS
```

Checker output establishes only conformance of supplied JSON bytes to the declared algorithm. The string `VALID` does not become an external truth.

21. Self-Application

This manuscript is an exact record. Its title, theorem strings, formulas, and the words `полностью`, `источник`, and `цикл`—read as “fully,” “source,” and “cycle”—receive no exemption.

Status layers are separated by exact protocol fields. The title and three theorem claims use separate records, not a `claim_ids` field:

```
claim_id: C-PUBLIC-TITLE
kind: title
document: 00_MANUSCRIPT_RU.md
expected_status: UNBOUNDED-STANDALONE
semantic_status: NOT-MACHINE-VERIFIED

claim_id: C-THEOREM-CP-LFP-3
kind: textual_theorem
document: 01_CYCLIC_PROVENANCE_THEOREM_RU.md
expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY
semantic_status: NOT-MACHINE-VERIFIED

claim_id: C-THEOREM-SG-LFP-3
kind: textual_theorem
document: 02_SUPPORT_GROUNDING_THEOREM_RU.md
expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY
semantic_status: NOT-MACHINE-VERIFIED

claim_id: C-THEOREM-FP-SEL-3
kind: textual_theorem
document: 03_FIXED_POINT_SELECTION_RU.md
expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY
semantic_status: NOT-MACHINE-VERIFIED
```

These exact RU paths are source-record objects; the EN companion does not replace or inherit their self-audit status. Each text record also contains `exact_text`, `exact_text_sha256`, and `claim_sha256`. Each artifact record uses `singular claim_id`, `kind: bounded_artifact`, `artifact`, `artifact_sha256`, `checker`, `expected_reason_codes`, `expected_observation`, `claim_sha256`, and:

```
expected_artifact_status: VALID-CYCLIC-PROVENANCE-LFP
                        | INVALID-CYCLIC-PROVENANCE
                        | VALID-SUPPORT-GROUNDING-LFP
expected_status: VERIFIED-IN-DECLARED-ARTIFACT
semantic_status: NOT-MACHINE-VERIFIED
```

Here `expected_artifact_status` is a self-audit-manifest field; top-level runtime `status` occurs in the checker result. Wildcards `VALID-*`, `INVALID-*`, and `INCOMPATIBLE-*` are not literal enum values in the self-audit schema.

A manuscript reference to its own provedness forms an edge/payload of this record. It introduces no theorem status. If two reading contracts contain identical manuscript text but different

`applies_to_all_mathematics` fields, the manuscript selects neither field. A later endorsement is a new descendant record.

22. Predecessors and the Novelty-Status Record

No external author, school, term, or adjacent work is a premise of CP-LFP-3, SG-LFP-3, or FP-SEL-3. Their derivations use objects declared inside the package.

The strings `least fixed point`, `SCC`, `provenance`, and `support` confer no historical priority through their presence. The schema-bound novelty record contains only:

```
claim_id: C-NOVELTY
kind: novelty_status
novelty_status: NOT-ASSESSED
priority_status: NO-PRIORITY-CLAIM
expected_status: NOT-ASSESSED
semantic_status: NOT-MACHINE-VERIFIED
claim_sha256: sha256:6d6a54955d87d6cd35a15cb50eb4a7e3f1c20d61a50cbdeb5f30518db9f9914d
```

`PREDECESSORS-AS-PREMISES: NONE` and `INTERNAL-PHASE-CONTRIBUTION: CP-LFP-3 + SG-LFP-3 + FP-SEL-3` are manuscript annotations, not fields of `phase3_self_claims.schema.json`. They do not change `novelty_status` OR `priority_status`.

The literal `NOT-ASSESSED` is a protocol string. It establishes neither novelty nor its absence. A later comparative record may change a historical label; it does not become a hidden premise of the derivation presented here.

23. Boundary of the Claims

Recorded Within the Declared Metatheory

1. In a finite positive-union graph, least provenance equals the set of origin-tagged introductions having a finite path to a node.
2. A source-free SCC does not generate a protected atom in the least fixed point by circulation alone.
3. An SCC with a local/upstream protected introduction transfers the atom while retaining its origin tag.
4. SCC condensation preserves reachability of introductions and localizes local/upstream sources.
5. Under the protected-endpoints internal-cut contract, an atom disappears from least root provenance after deletion iff deleted internal nodes intersect every path from every reaching introducer to the root.
6. In a finite positive support hypergraph, after normalization of nullary constructors and with nonempty premises for ordinary rules, every least-closure root has a finite support tree with seed leaves.
7. A pure self/mutual claim cycle without a seed and without an alternative finite seed-grounded support tree does not form a grounded root in least closure.

8. The support-grounded fixed point of T_S is unique and equals μT_S ; a nonleast fixed point contains unsupported circular residue. Unary Φ_G has the separate path-grounded consequence.
9. A fixed-point policy that changes the selected state under identical equations is a distinguishing input.
10. CP-AST extends the Phase 1 projection boundary to least cyclic provenance.

Not Supplied by This Package

- the line that all cycles are useless or inadmissible;
- the line assigning falsity to nonleast, greatest, stable, or coinductive fixed points;
- the line assigning falsity to every formal theorem;
- the line that no sign ever refers to anything;
- an event supplied by an event code;
- a referent supplied by a referent code;
- physical causation supplied by a cause code;
- justification supplied by a goal code;
- meaning supplied by a meaning code;
- exhaustiveness of the declared graph/readset over anything outside the record;
- correctness of a fixed-point policy outside its own contract;
- coverage of negation, defaults, defeat, probabilistic recursion, or temporal recursion by the positive-union theorem;
- an attested physical history supplied by a mutable or backdated record;
- absence of every future channel;
- a derivation of the historical origin of mathematics from language;
- a world-priority or global-novelty status for Phase 3.

This boundary does not relax the theorem. It prevents a cycle from receiving a privilege exactly where origin is absent, and prevents the title from acquiring an undeclared quantifier.

24. Conclusion

A closed record may be computationally complex. A recursive program may work. A formal system may contain self-reference. Two checkers may check one another. A controller and telemetry may form a feedback loop. Fixed-point semantics may be mathematically exact under its declared rules.

But in least provenance, a protected atom does not arise because a line returns to itself. It is present wherever a path leads from its introduction. With no introduction, the empty loop remains empty. If a

selected fixed point contains more, the selection rule is an added input. If $q \in \mu T_s$, a finite seed-grounded support tree exists; acceptance of a root under another policy is outside that statement.

The autonomous cyclic claim refuted is:

A closed record does not become the first source of a missing protected atom merely because its elements refer to one another.

Conditional recursive inferences are not rejected. The substitution of self-generation for transfer of an introduced atom is.

A cycle may have no first node.

An atom must still have an introduction.

Source Status

The work continues the internal beforeword line and Phases 0–2 of the study of the mathematical record boundary. External comparative sources are not premises. Phase 3 contains no field assigning historical priority or global novelty.

CP-LFP-3 (CFP-3): Least-Provenance Theorem for a Cyclic Record

0. Status of the notation

CP-LFP-3 is the normative theorem identifier. The shorter identifier CFP-3 is used as an equivalent alias. It expands to **Cyclic Provenance — Least Fixed Point**.

Every object below is an exact element of the declared formal apparatus. The lines `FORMAL`, `BINDING`, `EVENT`, `GOAL`, `REFERENT`, `CAUSE`, `APPLICATION`, and `MEANING` are uppercase mathematical sort codes; their lowercase JSON representatives are fixed in § 2.2. Their presence does not present a formula, binding, event, goal, referent, cause, application, or meaning outside the record.

The theorem does not use what is named as a premise. It uses only:

- equality of codes;
- membership in finite sets;
- recorded directed edges;
- protected introduction constructors;
- the least fixed point of an explicitly specified monotone operator.

`payload` is not a source of typed provenance. A string in `payload` that matches an atom id, sort name, serialized introduction, or the full text of this file adds nothing to the protected field `local_seeds`.

This English file is a companion written surface. Exact status and protocol lines remain source-record objects; their appearance here does not execute the Russian technical checker or transfer its result strings to this file.

1. Relation to Phases 0–2

Phase 0 fixed the non-erasure of a registered bridge dependency in an append-only trace. Phase 1 established the projection boundary: a function of the same complete readset cannot return two different required target codes. Phase 2 localized a distinguishing typed atom in the ancestry of a finite DAG.

CP-LFP-3 removes only one Phase 2 restriction: the record graph may contain directed cycles. Recursive provenance computation is replaced with an explicitly selected least fixed point. The remaining boundaries are not weakened:

- provenance remains origin-tagged;
- the target is fixed separately;
- the complete readset belongs to the base projection;

- a typed record does not become its referent;
- a provenance edge does not become a physical cause.

On a DAG, CP-LFP-3 reduces exactly to the Phase 2 ancestor union.

2. Finite cyclic record

2.1. Graph

Let

$$G = (V, E)$$

be a finite directed graph. Self-loops and directed cycles are permitted. An edge

$$u \rightarrow v$$

means only that the record u belongs to the declared dependency readset of node v . It does not mean physical transmission, causal action, temporal precedence, or semantic dependency.

Each node has the following protected fields:

- `node_id`;
- `kind`;
- a list of parents;
- `local_seeds` — the executable name for buckets of atom ids belonging to protected introductions; the buckets themselves are untagged, and an origin tag is added by pairing them with `node_id`;
- `declared_fixed_point` — the executable untagged projection of the claimed least provenance for that node;
- arbitrary payload.

CP-LFP-3 does not require a `birth_rank`: strict rank growth along every edge is impossible in a cycle. Physical time does not follow from the absence of a rank either.

2.2. Sorts and atom ids

Phase 3 uses the extended finite registry T_3 : exactly the following eight protected sorts:

- *FORMAL, BINDING, EVENT, GOAL*;
- *REFERENT, CAUSE, APPLICATION, MEANING*.

The JSON field `sort` represents these eight mathematical codes by the fixed bijection `FORMAL` $\leftrightarrow$ "formal", `BINDING` $\leftrightarrow$ "binding", `EVENT` $\leftrightarrow$ "event", `GOAL` $\leftrightarrow$ "goal", `REFERENT` $\leftrightarrow$ "referent", `CAUSE` $\leftrightarrow$ "cause", `APPLICATION` $\leftrightarrow$ "application", and `MEANING` $\leftrightarrow$ "meaning". The separate node field `kind` uses uppercase literal values and also permits `RELAY` and `VERDICT`.

The path theorem is parametric in a finite sort registry. The published Phase 3 executable contract enumerates exactly eight lowercase JSON representatives of the elements of T_3 . Its machine status applies only to the supplied finite JSON record and does not transfer to another sort because a name matches.

The first four sorts retain their Phase 2 roles. The additional sorts are also only sorts of records. In particular:

- REFERENCE is a referent-code record, not a referent;
- CAUSE is a cause-code record relative to a declared contract, not a physical cause;
- APPLICATION is an application-code record, not a fact of application;
- MEANING is a meaning-code record, not meaning;
- GOAL is a goal-code record, not the justification or acceptance of a goal;
- EVENT is an event-code record, not an event.

For each sort τ , let A_τ be a finite registry of atom ids. For each node v :

$$Intro_\tau(v) \subseteq A_\tau$$

is the finite set of atom ids introduced by the protected constructor for that sort.

A compatible contract must separately check the permitted node kind / introduced sort pairs. Arbitrary text in a node's payload does not authorize that node to introduce every sort.

2.3. Origin tags

The local origin-tagged introduction is

$$OIntro(v) = \{(v, \tau, a) : \tau \in T_3, a \in Intro_\tau(v)\}.$$

The complete finite registry of possible origin-tagged atoms is

$$\Omega = V \times \bigcup_{\tau \in T_3} (\{\tau\} \times A_\tau).$$

The triples (v, τ, a) and (w, τ, a) are distinct when $v \neq w$. A string depicting such a triple in payload is not a protected origin tag.

A checker that propagates only untagged pairs (τ, a) checks the untagged projection of the path theorem. To receive the status of complete origin-tagged provenance, it must either propagate triples (s, τ, a) at every node or exhaustively reconstruct and supply every introducer-node id for every admitted atom at every checked node. One origin witness for one query is not the complete origin-tagged closure.

3. Cyclic-provenance operator

Consider the finite complete lattice

$$L = (2^{\mathcal{Q}})^V$$

ordered by coordinatewise inclusion. An element $X \in L$ assigns a set of origin-tagged atoms $X(v)$ to each node.

The CP-LFP-3 operator is

$$\Phi_G(X)(v) = OIntro(v) \cup \bigcup_{u: u \rightarrow v} X(u).$$

It is monotone:

$$X \subseteq Y \Rightarrow \Phi_G(X) \subseteq \Phi_G(Y).$$

Normative provenance is defined not by an arbitrary solution of $X = \Phi_G(X)$, but by iteration from the bottom:

$$P^0(v) = \emptyset, P^{n+1} = \Phi_G(P^n),$$

$$OProv_G^\mu = \mu \Phi_G = \bigcup_{n \geq 0} P^n.$$

Because V is finite and new atoms are only unioned, the iteration stabilizes. The transition $P^0 \rightarrow P^1$ inserts local introductions; no more than $|V| - 1$ further state-changing propagation transitions are then needed, because every reachability relation has a simple path of length at most $|V| - 1$. Thus $P^{|V|} = \mu \Phi_G$, although an implementation may need one additional equality check to detect stabilization.

The checker must compute exactly this least closure. The executable implementation initializes `current` with the untagged projection

$$P^1(v) = OIntro(v),$$

omitting deterministic storage of P^0 and the transition $P^0 \rightarrow P^1$. Accordingly, `iteration_count` counts only subsequent transitions that strictly change at least one untagged bucket; it does not count the final unchanged check. This is a runtime counter, not the index n in the proof, and the offset does not change $\mu \Phi_G$. Accepting any other fixed point as `declared_fixed_point` is incompatible with CP-LFP-3.

4. CP-LFP-3: path theorem

The notation

$$s \rightsquigarrow_G^* v$$

means that a finite directed path exists from s to v , including a path of length zero when $s = v$.

The theorem quantifies over the existence of a path and does not select its serialization. The executable verdict separately supplies one deterministic shortest-lex simple path in `query_results[].path_witnesses[].node_ids`: path length is minimized first, and the `node_id` sequence is compared lexicographically second. This witness policy is not an additional premise of the path equivalence.

Theorem CP-LFP-3

For every $v \in V$:

$$OProv_G^\mu(v) = \{(s, \tau, a) : a \in Intro_\tau(s) \wedge s \rightsquigarrow_G^* v\}.$$

Thus least cyclic provenance contains exactly the protected origin-tagged introductions that reach the node along finite recorded paths.

Proof

For $n \geq 1$, define

$$R_n(v) = \{(s, \tau, a) : a \in Intro_\tau(s), \text{существует путь } s \rightsquigarrow_G^* v \text{ длины } < n\}.$$

Here the exact formula strings `существует путь` and `длины` have the secondary English glosses `there is a path` and `of length`.

The equality $P^n(v) = R_n(v)$ follows by induction.

For $n = 1$:

$$P^1(v) = OIntro(v) = R_1(v),$$

because the only paths of length less than one are reflexive paths of length zero.

Assume $P^n(u) = R_n(u)$ for every u . Then

$$P^{n+1}(v) = OIntro(v) \cup \bigcup_{u: u \rightarrow v} R_n(u).$$

The first term gives paths of length zero. The second gives exactly those paths into v that end with an edge $u \rightarrow v$ and have length less than n before reaching u . It therefore gives the positive-length paths of length less than $n + 1$. Hence

$$P^{n+1}(v) = R_{n+1}(v).$$

Taking the union over all n gives all and only finite paths. In a finite graph, every reachable node has a simple path, so stabilization is finite. $\square$

Sharp consequence

If an origin-tagged atom belongs to $OProv_G^\mu(v)$, there is an exact node s at which that atom is introduced by a protected constructor, together with a finite path $s \rightsquigarrow_G^* v$.

The converse is exact as well: every protected introduction with a path to v belongs to the least provenance of v .

No part of the theorem establishes that an atom id presents what it names.

5. A cycle is not a source

5.1. Exact no-source condition

For $C \subseteq V$, define the incoming cone by

$$InCone(C) = \{s \in V : \exists v \in C, s \rightsquigarrow_G^* v\}.$$

For a fixed atom code (τ, a) , NO-UPSTREAM-INTRO means

$$\forall s \in InCone(C) : a \notin Intro_\tau(s).$$

The introduction must be absent throughout the incoming cone, not merely inside C . A cycle with no local introduction may legitimately propagate an atom arriving along an incoming path.

Corollary CP-NOSOURCE

If NO-UPSTREAM-INTRO holds, then for every $v \in C$:

$$(s, \tau, a) \notin OProv_G^\mu(v) \text{ для всех } s.$$

Here the exact formula string *для всех* has the secondary English gloss *for every*.

This follows immediately from CP-LFP-3.

The exact Russian public line is:

Цикл может переносить введённую запись. Цикл не является её введением.

Secondary English gloss:

A cycle can propagate an introduced record. A cycle is not its introduction.

The gloss is a separate written surface and does not replace the source line.

5.2. Positive control

Let $v_0 \rightarrow v_1 \rightarrow v_0$, while a node s outside the component introduces (τ, a) and has an edge $s \rightarrow v_0$. The atom then belongs to the least provenance of both cycle nodes. This is not a counterexample: s is the source, and the cycle only propagates the introduction.

If v_0 introduces the atom locally, it also reaches its entire strongly connected component. The local introducer is the source; closure of the path is not itself a source.

6. SCC condensation

Let $SCC(G)$ be the partition of V into strongly connected components. The condensation

$$D_G = G / SCC(G)$$

is a DAG. For a component C , define

$$J(C) = \bigcup_{s \in C} OIntro(s).$$

Theorem CP-SCC

For a node v in component C_v :

$$OProv_G^\mu(v) = \bigcup_{C: C \sim_{D_G}^* C_v} J(C).$$

Proof

A directed path between nodes induces a path between their SCCs in the condensation. Conversely, a component path can be lifted to a path between the required nodes, while all nodes inside each SCC are mutually reachable. A source node therefore reaches v in G exactly when its component reaches C_v in D_G . Substituting this reachability equivalence into CP-LFP-3 gives the formula. $\square$

Condensation removes repeated traversal of a cycle while preserving every origin-tagged introduction. It gives the exact role of an SCC: propagating incoming and local introductions throughout its reachability region.

Recovery of Phase 2

If G is a DAG, every SCC is a singleton. CP-SCC then becomes

$$OProv_G^\mu(v) = \bigcup_{u \in Anc_G^*(v)} OIntro(u),$$

which is the Phase 2 IADC/ancestor union.

7. A cut in a cyclic graph

Write

$$Prov_H^\mu(x) = \{(\tau, a) : \exists s \in V(H) (s, \tau, a) \in OProv_H^\mu(x)\}.$$

Let $r \in V$ be a protected root. For a fixed untagged code (τ, a) , the set of introducers reaching the root is

$$S_{\tau,a}^r = \{s \in V : a \in Intro_\tau(s) \wedge s \rightsquigarrow_G^* r\}.$$

Let $P_G(s, r)$ denote the set of finite directed paths from s to r , and let $V(p)$ be the vertex set of path p .

Under the internal-cut contract, all introducers in $S_{\tau,a}^r$ and the root are protected. Let

$$B \subseteq V \setminus (S_{\tau,a}^r \cup \{r\}).$$

The blocking operation removes the internal nodes B and their incident edges, performs no rewiring, and then recomputes the least fixed point on $G - B$.

Theorem CP-CUT

After blocking, the untagged atom (τ, a) is absent from the provenance of r exactly when B intersects every directed path from every $s \in S_{\tau,a}^r$ to r :

$$(\tau, a) \notin Prov_{G-B}^\mu(r) \Leftrightarrow \forall s \in S_{\tau,a}^r \forall p \in P_G(s, r), V(p) \cap B \neq \emptyset.$$

Proof

By CP-LFP-3, the atom remains at the root after blocking exactly when a path survives from a protected introducer to r . The negation of such a path's existence says that B intersects every introducer-to-root path. Cycles do not alter the criterion: every walk reaching the root contains a simple path with the same endpoints. $\square$

If $S_{\tau,a}^r = \emptyset$, the nested `cut_status` is `ABSENT` and `rank` is `null`; an empty hitting set is not serialized as a cut certificate. If an introducer is the root or has a direct edge to the root, no internal cut exists: protected endpoints cannot be removed, and `cut_status` is `UNBLOCKABLE-PROTECTED-ENDPOINT`. In every other non-empty case, a minimum-cardinality B is a finite combined internal vertex-cut certificate with status `CUT-CERTIFIED`. Its size measures only the structural redundancy of the recorded provenance routes. It does not measure truth, physical causal force, utility, or the meaning of an atom code.

8. Composition with AST-1

Let $K \neq \emptyset$ be a declared class of exact-record indices. The base projection

$$\rho: K \rightarrow R$$

must include or uniquely fix:

- the complete G , node kinds, and root v ;
- the sort and atom-id registries;
- the structure, registries, and type policy of introductions; exact introduction values that form the root provenance are assigned to η_v^μ and are not duplicated in ρ when the evaluator reads them in no other way;
- the operator Φ_G and the `least` selection rule;
- the evaluator contract;
- every seed, transcript, schedule, oracle output, or other input actually read.

If any read component differs, ρ must differ as well.

The cyclic-provenance view is

$$\eta_v^\mu(k) = \text{canon}\left(\text{OProv}_{G_k}^\mu(v)\right),$$

where `canon` is injective on exact finite sets of origin-tagged triples.

Let $Q: K \rightarrow Y$ be an independently fixed target coordinate, and let

$$C_v(k) = F(\rho(k), \eta_v^\mu(k))$$

for a total function F . If, throughout K ,

$$C_v(k) = Q(k),$$

then

$$\ker \rho \cap \ker \eta_v^\mu \subseteq \ker Q.$$

Proof

If $\rho(k_0) = \rho(k_1)$ and $\eta_v^\mu(k_0) = \eta_v^\mu(k_1)$, the function F receives the same argument and returns the same output code. The equality $C_v = Q$ carries that equality to Q . ◻

Origin witness under cycles

If $\rho(k_0) = \rho(k_1)$ but $Q(k_0) \neq Q(k_1)$, then, under a correct export,

$$\eta_v^\mu(k_0) \neq \eta_v^\mu(k_1).$$

The injectivity of canon gives an origin-tagged triple in the symmetric difference. By CP-LFP-3, where that triple is present it has a protected introducer and a finite introducer-to-root path. The path may pass through an SCC; the SCC does not replace the introducer.

This extends the structural localization of AST-1 to cyclic records. It does not turn the target code $Q(k)$ into a referent, event, cause, application, goal, or meaning.

9. Complete closure does not create an external target

The view (ρ, η_v^μ) may include:

- the entire cyclic graph;
- every iteration from $\perp$;
- the finite least fixed point;
- every formal payload;
- proof traces;
- self-reference strings;
- controller states and feedback transcripts, when recorded.

If two admissible indices have the same complete view while independently specified acceptable target sets $A(k_0), A(k_1) \subseteq Y$ have an empty intersection, no zero-error selector from that view exists:

$$(\rho, \eta_v^\mu)(k_0) = (\rho, \eta_v^\mu)(k_1), A(k_0) \cap A(k_1) = \emptyset$$

implies

$$\nexists f \forall i \in \{0, 1\} : f(\rho(k_i), \eta_v^\mu(k_i)) \in A(k_i).$$

This is a direct application of RBI-1/FDT-2 after the cyclic closure has been computed. A cycle does not split a fiber when the complete cyclic view actually read is identical.

The condition is checked separately for each exact sort `FORMAL`, `BINDING`, `EVENT`, `GOAL`, `REFERENT`, `CAUSE`, `APPLICATION`, and `MEANING`. Here, `purpose` and `justification` may be only separately typed claim-role codes, not implicit additional sorts. Successful selectability for one target is not inherited by another target.

10. Covariance under renaming

Let π be a total bijection that consistently renames:

- node ids;
- sort ids;

- atom ids;
- graph edges;
- protected introductions;
- all corresponding operator and evaluator-contract fields.

Partial renaming of payload without transporting the rules is not covered.

For the transported graph πG :

$$\pi(OProv_G^\mu(v)) = OProv_{\pi G}^\mu(\pi v).$$

Proof

The bijection preserves and reflects introductions and finite reachability. Applying CP-LFP-3 on both sides yields the same transported triples. Equivalently, $\pi \circ \Phi_G = \Phi_{\pi G} \circ \pi$, and the lattice isomorphism transports the least fixed point. $\square$

The shape of a glyph and the name `EVENT` therefore have no privilege within the theorem. If the output changes under a complete, structurally consistent recoding, the premise of complete transport has been violated. If only the external assignment changes while the formal view remains fixed, a separate binding channel is required.

11. Snapshots and the exact boundary of non-retroactivity

Phase 2 prohibited new edges into old nodes by means of forward birth rank. The cyclic contract does not contain that prohibition automatically.

Let (G, I) be an old snapshot and (G', I') an extended snapshot that preserves old nodes, edges, and introductions but permits new edges. For an old root r :

$$OProv_{G'}^\mu(r) \setminus OProv_G^\mu(r)$$

contains exactly those introductions for which G' has a path to r that did not exist in G , including introductions at new nodes.

Old provenance therefore remains unchanged exactly when the extension creates no new introducer-to-root path. Adding a back edge may change the closure of an old root in the new snapshot. It does not rewrite the bytes of the old snapshot, but it removes any unconditional claim of non-retroactivity for the new graph.

Compatible fixation strategies include:

1. an immutable versioned snapshot: the old root is computed only in the old G ;

2. a prohibition on new paths to protected old roots;
3. a new root/version for the new closure.

A time string or hash is not physical time by itself.

12. Admission gates

An instance receives CP-LFP-3 status only when all of the following hold:

1. V, E, T_3, A_τ , and the introductions are finite and structurally valid;
2. every parent id resolves;
3. the operator is exactly (CP), or a predeclared equivalent union operator;
4. the operator is monotone on the finite powerset domain;
5. provenance is computed by iteration from $\perp$, not selected as an arbitrary fixed point;
6. each untagged bucket `declared_fixed_point` is exactly the untagged projection of the computed $OProv_G^\mu$ at the corresponding node;
7. each origin tag is bound to an actual `node_id` and a protected introduction of the corresponding sort;
8. payload is not read as an introduction;
9. the complete evaluator readset is included in ρ ;
10. the target and acceptable sets are fixed independently of the output.

Non-monotone recursion, negation as failure, and atom deletion receive `OUTSIDE-CP-LFP-3-NONMONOTONE`. A hidden clock, scheduler, oracle, or mutable state defeats completeness of the readset; an infinite instance lies outside the finite executable contract.

The general set-theoretic path formula also permits an infinite graph for the union operator, but the executable Phase 3 contract is restricted to finite records. A machine status does not transfer to an infinite object.

13. Normative statuses

The checker's literal top-level schema statuses are, for an instance verdict, `VALID-CYCLIC-PROVENANCE-LFP`, `INVALID-CYCLIC-PROVENANCE`, and `INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT`; and, for a contract-audit verdict, `CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS` and `CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL`. `ABSENT`, `CUT-CERTIFIED`, and `UNBLOCKABLE-PROTECTED-ENDPOINT` are values of the nested field `query_results[].minimum_internal_vertex_cut.cut_status`, not of the top-level status. The remaining table entries are protocol-level labels for theorems and controls.

These are source protocol strings. Their presence in the EN companion does not execute the source checker or inherit its verdict.

Status	Exact condition	What the status does not establish
VALID-CYCLIC-PROVENANCE-LFP	every declared_fixed_point equals the untagged projection of $OProv_G^\mu$	truth of a payload or target
CYCLE-PROPAGATES-INTRODUCED	an atom has an upstream/local introducer and a path through an SCC	that an atom id presents what it names
NO-ORIGINLESS-ATOM-IN-LFP	an introducer and finite path are found for every atom in least provenance	absence of an event, goal, or meaning outside the record
SCC-CONDENSATION-PRESERVES-PROVENANCE	CP-SCC holds on the exact instance	physical equivalence of systems
ABSENT	no introducer for the query atom reaches the root	absence of what is named outside the record
CUT-CERTIFIED	an exact minimum combined internal cut separates all protected sources from the root	physical causality of cut nodes
UNBLOCKABLE-PROTECTED-ENDPOINT	a local/direct protected path has no internal cut	non-erasure of the atom beyond this contract
CYCLIC-AST-SEPARATED	a target conflict is accompanied by a difference in η_v^μ	correctness of the target assignment outside the contract
PAYLOAD-SPOOF-REJECTED	a string is not admitted as a protected introduction	falsity of the string
OUTSIDE-CP-LFP-3-NONMONOTONE	the operator is non-monotone	a negative result about another operator

Status names remain protocol strings.

14. Required controls

1. **Empty self-loop.** $v \rightarrow v$, with no introductions. Least provenance is empty.
2. **Two mutually connected nodes without a source.** $u \leftrightarrow v$, with no introductions. Least provenance is empty at both nodes.
3. **Incoming source.** s introduces an atom and points into $u \leftrightarrow v$. The atom reaches both cycle nodes.
4. **Local source.** One SCC node introduces an atom. The atom reaches the entire SCC and every downstream node.
5. **Payload spoof.** A self-loop contains the line `EVENT: flight occurred`, but `local_seeds.event=[]`. Event provenance is empty.
6. **Origin spoof.** `declared_fixed_point.event` contains e at a node with no reachable `local_seeds.event` introducer. The instance is rejected.

7. **SCC collapse.** Provenance at every root agrees before and after condensation, with origin tags preserved.
8. **Cyclic cut.** With two routes through a cycle, an incomplete cut preserves the atom; a hitting set of all simple paths removes it.
9. **Back-edge update.** A new introducer receives a path to an old root: the new snapshot receives the atom, while the old versioned snapshot does not change.
10. **Complete recoding.** A complete bijective renaming commutes with least closure.
11. **Partial recoding.** Renaming the carrier without transporting the parser/rules is not issued as CP-RECODE.
12. **Completion conflict.** The complete cyclic view is identical and singleton target codes differ: no autonomous selector exists.
13. **Overlap control.** Acceptable sets differ but share a code: the shared zero-error code is admissible.
14. **Constant target.** The target is constant on a view fiber: factorization is admissible.

Positive controls 3, 4, 13, and 14 are mandatory: the theorem must not be implemented as an unconditional refusal to admit a cycle, propagation, or a selector.

15. Exact falsifiers

The following falsifiers are mathematical countermodels when the admission gates remain fixed. A checker's mere acceptance of an erroneous JSON record is not yet such a countermodel.

CP-PATH-FALSIFIER

CP-LFP-3 is defeated on an exact instance if, with every admission gate preserved, at least one of the following is supplied:

1. an origin-tagged atom belongs to the computed $OProv_G^\mu(v)$, but there is no corresponding protected introducer and finite introducer-to- v path;
2. a protected introducer has a finite path to v , but the corresponding origin-tagged atom is absent from $OProv_G^\mu(v)$.

CP-SCC-FALSIFIER

Condensation preserves exact introductions and reachability between components, but the CP-SCC formula yields different root provenance.

CP-CUT-FALSIFIER

After internal nodes are removed without rewiring, introducers and the root remain protected, and the least closure is recomputed:

- every introducer-to-root path is intersected, but the atom remains; or

- a surviving introducer-to-root path exists, but the atom disappears.

CP-AST-FALSIFIER

There are k_0, k_1 with identical ρ and η_v^μ , and different Q , while a total function F in fact receives only those two arguments and satisfies $F(\rho, \eta) = Q$ throughout the declared K .

A hidden oracle, different scheduler, incomplete ρ , modified graph, different fixed-point policy, post hoc target, or semantic assertion without a record code defeats a premise; it is not a falsifier.

Separately: executable-conformance defect

If the checker accepts a `declared_fixed_point` different from the untagged projection of the computed $OProv_G^\mu$, extracts an introduction from `payload`, accepts a prohibited kind/sort pair, supplies an incorrect origin/path witness, or conflates a nested cut status with top-level `status`, this is an executable-conformance defect. It defeats the claim that the implementation conforms to the contract. It defeats CP-LFP-3 only when a separate mathematical counterexample to the path equivalence is supplied.

16. Self-application and the maximal line

This file is itself a carrier. The presence of the line `CP-LFP-3` does not create a theorem outside the selected reading; the presence of the line `MEANING` does not create meaning; the presence of a proof does not present an external referent. Within the declared metatheory, the written sequence derives a conditional path formula. The same boundary applies to the document itself.

The exact Russian maximal formulation of Phase 3 is:

В наименьшем монотонном происхождении циклической записи каждый защищённый атом у root имеет точный introducer и конечный путь от него. Strongly connected component может циркулировать локальное или входящее введение, но не становится его источником. Выбор произвольной nonleast fixed point, скрытая policy или внешний target требуют отдельной записи и не возникают из повторения прежнего carrier.

Secondary English gloss:

In the least monotone provenance of a cyclic record, every protected atom at the root has an exact introducer and a finite path from it. A strongly connected component can circulate a local or incoming introduction, but does not become its source. Selecting an arbitrary non-least fixed point, a hidden policy, or an external target requires a separate record and does not arise from repetition of the earlier carrier.

The gloss is a separate written surface and does not replace the source line.

The record does not claim:

- that every formal derivation is false;
- that cycles physically do nothing;
- that a referent, event, cause, application, purpose, or meaning is absent;
- that the declared graph is a physical causal system;
- that the least fixed point is the only admissible semantics for every other contract;
- that a status code authenticates what it names.

SG-LFP-3 (GST-3): Support and Protected-Introduction Theorem

0. Normative status

SG-LFP-3 is the normative identifier. GST-3 is the short alias: **Grounded Support Theorem**. This file extends the unary propagation of CP-LFP-3 to finite positive AND/OR rules.

Here the words `seed`, `support`, `rule`, `root`, `proof`, and `grounded` denote fields and relations of an exact-record contract. They do not authenticate a ground outside the record, truth, observation, cause, goal, referent, or meaning. In particular:

- `seed` is a protected nullary introduction of a record code;
- `support tree` is a finite tree of rule ids and code ids;
- `grounded` means that such a tree exists;
- `root` is the selected query code id.

The exact Russian line `это основание` (secondary gloss: this is a ground) in `payload` does not create a seed. The exact Russian line `из А следует В` (secondary gloss: B follows from A) does not create a hyperrule. Both objects are registered by protected constructors.

This English file is a companion written surface. Exact protocol and status strings below remain source-record objects; their presence here neither executes the Russian technical checker nor transfers its result strings.

1. Why an AND/OR layer is needed

CP-LFP-3 treats propagation of one identical origin-tagged atom along edges. A formal record often has a more general form:

- one output requires several premise codes simultaneously — AND;
- one output may be obtained by different rule ids — OR;
- rule dependencies may form cycles;
- one root may have several incompatible or overlapping seed bases.

SG-LFP-3 addresses the exact question:

When does a root code belong to the least closure of a finite positive rule record, and which finite set of protected seeds supports that membership?

The answer does not use a semantic reading of a code id. It uses only membership, subset, and finite iteration.

2. Finite support contract

2.1. Abstract code universe

Let N be a finite non-empty set of exact code ids. Elements of N may have the typed form

$$n = (v, \tau, a),$$

where v is an origin node, τ a sort id, and a an atom id. The theorem does not require this decomposition and applies to arbitrary exact code ids.

Let

$$S \subseteq N$$

be the set of protected seed/base codes. In the abstract theorem, a seed may also be a code that participates in rules. In the executable refinement below, the base and claim namespaces are separated by tagged embedding. Matching payload under a different protected id does not merge records.

2.2. Positive hyperrules

The finite rule set R consists of records

$$r = \langle id_r, Prem(r), Head(r) \rangle,$$

where

$$\emptyset \neq Prem(r) \subseteq N, Head(r) \in N.$$

All premise codes of one rule are required simultaneously. Different rules with the same $Head$ are alternatives.

The normative domain condition after normalization is

$$Prem(r) \neq \emptyset$$

meaning

$$\forall r \in R_{ordinary} Prem(r) \neq \emptyset.$$

A nullary constructor $\emptyset \rightarrow h$ is not silently included in $R_{ordinary}$. Before the theorem is applied, it is normalized into a fresh protected base $b_r \in S$ and the ordinary rule $\{b_r\} \rightarrow h$. One admissible convention id for b_r is:

RULE-SEED:<rule-id>

2.3. Exact executable embedding

Let

$$B = \{(BASE, b) : b \text{ — declared 'base_id'}\}, C = \{(CLAIM, c) : c \text{ — declared 'claim_id'}\}.$$

The exact formula string — declared is already English; the backticked field names remain exact source literals.

The tags make $B \cap C = \emptyset$, even if the same raw string is permitted in both namespaces. The executable record is embedded in the abstract system with $N = B \sqcup C$, complete seed set $S = B$, and each inference r specifying

$$\begin{aligned} id_r &= inference_{id}, Prem(r) = P_B(r) \cup P_C(r). \\ P_B(r) &= \{(BASE, b) : b \in \text{required_base_ids}\}. \\ P_C(r) &= \{(CLAIM, c) : c \in \text{premise_claim_ids}\}. \\ Head(r) &= (CLAIM, conclusion_{id}). \end{aligned}$$

This embedding preserves the AND of all base/claim premises and the claim projection of the least closure. The complete executable seed set is $S = B$. When a candidate basis is checked, the new notation $B' \subseteq B$ is used: B' replaces S as the operator's seed argument, while N and R remain unchanged. A normalized nullary constructor creates a protected base and an inference to the required conclusion_id, explicitly listing that base_id in required_base_ids. The checker does not itself normalize the input and does not require the prefix RULE-SEED; its exact executable domain condition is that the union of premise_claim_ids and required_base_ids be non-empty for every inference. If both arrays are empty, it returns status INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT and reason_codes=["NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE"].

2.4. Exact correspondence of runtime fields

Runtime field	Abstract object or exact effect
bases[].base_id = b	tagged code $(BASE, b) \in B = S$
bases[].role_code	checked metadata classification; copied into base_catalog but absent from the premise test
claims[].claim_id = c	tagged code $(CLAIM, c) \in C$
inferences[].inference_id	id_r
inferences[].conclusion_id	raw representative of head $(CLAIM, c)$

Runtime field	Abstract object or exact effect
<code>inferences[].premise_claim_ids</code>	claim part of $Prem(r)$
<code>inferences[].required_base_ids</code>	base part of $Prem(r)$
<code>claims[].declared_minimal_supports</code>	declared exact antichain of minimal subsets of raw <code>base_id</code> ; the checker requires its fixed-point equality and equality to the computed least frontier
<code>query.claim_ids</code>	non-empty projection of an already computed result; does not change closure
<code>claim_grounding[].grounded</code>	true exactly when the computed frontier of that claim is non-empty
<code>claim_grounding[].minimal_supports</code>	computed inclusion-minimal antichain of raw <code>base_id</code>
<code>claim_grounding[].producer_inference_ids</code>	every declared inference with that <code>conclusion_id</code> , not only the inference used by the selected witness
<code>support_tree_witnesses</code>	one deterministic witness for every claim/minimal-support pair
<code>ungrounded_claim_ids</code>	claims with an empty least frontier
<code>unseeded_cycle_claim_sets</code>	cyclic claim SCCs in which every claim has an empty least frontier
<code>iteration_count</code>	number of strict frontier-iteration transitions before stabilization
<code>artifact_sha256</code>	canonical digest of the entire accepted input record

2.5. Payload inertness and canonical digest

The runtime checks the `payload` of every base, claim, and inference as a permitted JSON value with `MAX_PAYLOAD_DEPTH = 32` (under the overall record depth bound of 64), but `infer_once` does not read payload as a premise, base, rule, or head. Rule support is determined only by protected ids and contract policy; a similar string inside `payload` is not extracted as a rule.

For an accepted record, `artifact_sha256` equals `"sha256:" + hex(SHA256(canonical_bytes(value)))` for the complete input value, including `contract`, `query`, and all payload values. Under the `bw-cjson-2-python-reference` profile, `canonical_bytes` is UTF-8 JSON with sorted object keys, separators `,` and `:` without additional whitespace, `ensure_ascii=false`, and non-finite numbers prohibited. This is a digest of the canonical value, not of the original whitespace or object-key order; Unicode is not normalized. Changing a canonical payload value may therefore change `artifact_sha256`, but with the protected fields fixed it does not change N , S , R , the least frontier, or support witnesses. Merely placing a complete JSON rule, base, tree, or result in payload does not create support.

3. Least AND/OR closure

On the finite lattice 2^N , define

$$T_S(X) = S \cup \{ \text{Head}(r) : r \in R, \text{Prem}(r) \subseteq X \}.$$

It is monotone:

$$X \subseteq Y \Rightarrow T_S(X) \subseteq T_S(Y).$$

Iteration from the bottom,

$$C_S^0 = \emptyset, C_S^{n+1} = T_S(C_S^n),$$

defines

$$Cl_R(S) = \mu T_S = \bigcup_{n \geq 0} C_S^n.$$

Because N is finite and the operator only adds codes, the iteration stabilizes after finitely many rounds.

ci means only membership in the least record closure. The line

$$q \in Cl_R(S)$$

does not mean that payload q is true or that what it names exists.

4. Finite support tree

4.1. Definition

A support tree for a root code $q \in N$ is a finite rooted tree D , each of whose nodes is labeled by a code id in N , such that:

1. the root is labeled q ;
2. every leaf is labeled by an element $s \in S$;
3. every internal node labeled h contains an exact rule id $i d_r$, where $\text{Head}(r) = h$;
4. the set of child labels is exactly $\text{Prem}(r)$, every premise is represented by exactly one child, and the serialization of children uses the contract's canonical order;
5. every rule id and code id resolves in the supplied record.

In the abstract tree, each seed leaf is labeled by some code $s \in S$. In the executable embedding, the complete seed set is $S = B$, so s has the tagged form (BASE, b) , while the raw string $\text{base_id} = b$ is only its compact representative after resolution in the `bases` table. The compact verdict does not create a separate JSON tree node for such a leaf: b is recorded in the parent claim node's `required_base_ids`, and the

set of all base leaves in that subtree is recorded in `support_base_ids`. `premise_trees` contains only claim children. In the executable embedding, an abstract rule id is represented by `inference_id`, and the head by a tagged `conclusion_id`. Thus, the absence of a separate JSON child node for a base neither removes the abstract leaf nor identifies the raw string with an element of S without a namespace tag.

The same code id may occur in different branches. The tree must be finite. An infinite branch ending with a reference to itself is not a support tree.

The set of abstract seed codes at the leaves is

$$Leaves(D) \subseteq S.$$

A repeated seed is counted once in a set-valued basis. If a contract requires resource multiplicity, it needs a separate multiset semantics, not SG-LFP-3.

4.2. Height

A leaf has height zero. An internal node has height one plus the maximum height of its children. Finite height is an operational witness that no infinite regress occurs inside the tree.

5. SG-LFP-3: support-tree theorem

Theorem SG-LFP-3

For every $q \in N$, the following are equivalent:

$$q \in \mu T_S;$$

существует конечное support tree для q с листьями в S .

The exact formula strings `существует конечное support tree для` and `с листьями в` have the secondary English glosses there is a finite support tree for and with leaves in.

In boxed form:

$$q \in Cl_R(S) \Leftrightarrow \exists D \text{ SupportTree}_{R,S}(D, q).$$

Proof of $(A) \Rightarrow (B)$

Let n be the first index for which $q \in C_S^{n+1}$.

If $q \in S$, the one-node tree whose leaf is q is a support tree. Otherwise, a rule r exists with $Head(r) = q$ and

$$Prem(r) \subseteq C_S^n.$$

For each premise $p \in \text{Prem}(r)$, its first-entry index is strictly smaller than the first-entry index of q . Induction on first-entry rank gives a finite support tree for every p . The new root q receives rule id r , and those trees are attached as children. This yields a finite support tree for q . $\square$

Proof of $(B) \Rightarrow (A)$

Proceed by induction on the height of the support tree. Every leaf belongs to $S \subseteq T_S(X)$ for every X .

Suppose every child of an internal node h belongs to the closure. Their finite set is $\text{Prem}(r)$ for a declared rule r with head h . At some common finite iteration all premises are present; the next iteration adds h . The root therefore belongs to μT_S . $\square$

Seed-ancestry consequence

Every root membership in the least closure has at least one finite support path to a seed code $s \in S$. A cyclic reference may occur inside the selected record, but no branch of an admitted support tree ends in a cycle: in the abstract theorem it ends in $s \in S$, and in the executable embedding that leaf is represented by an exact `base_id`.

This is structural seed ancestry, not historical, physical, or semantic ancestry.

6. Minimal root bases

6.1. Basis

A candidate $B' \subseteq S$ is a root basis for q when

$$q \in Cl_R(B'),$$

where the rules R are unchanged and the available seeds are restricted to the exact set B' .

A basis is inclusion-minimal when

$$q \in Cl_R(B')$$

and, for every proper $B'' \subsetneq B'$,

$$q \notin Cl_R(B'').$$

The family of all inclusion-minimal bases is denoted

$$B_{min}(q).$$

If $q \notin Cl_R(S)$, this family is empty.

Theorem SG-BASIS

$$B_{min}(q) = \min_{\subseteq} \{ Leaves(D) : D \text{ — support tree для } q \}.$$

The exact formula string — support tree для has the secondary English gloss — a support tree for.

Proof

Every support tree with leaf set L shows, by SG-LFP-3, that

$$q \in Cl_R(L),$$

so L is a basis.

If an inclusion-minimal leaf set L were not a minimal basis, a proper sub-basis $B \subsetneq L$ would exist. SG-LFP-3 for seed set B would give a support tree with leaf set $L' \subseteq B \subsetneq L$, contradicting the minimality of L among support-tree leaf sets. Every inclusion-minimal leaf set is therefore a minimal basis.

Conversely, let B be a minimal basis. SG-LFP-3 for seed set B gives a support tree D with $Leaves(D) = L \subseteq B$. If $L \subsetneq B$, then $q \in Cl_R(L)$, contradicting the minimality of B . Hence $L = B$. Basis minimality and an inclusion-minimal leaf set coincide. $\square$

6.2. May- and must-seeds

If $B_{min}(q) \neq \emptyset$, define

$$MaySeed(q) = \bigcup_{B \in B_{min}(q)} B,$$

$$MustSeed(q) = \bigcap_{B \in B_{min}(q)} B.$$

MaySeed lists seeds that occur in at least one minimal basis; MustSeed lists seeds that occur in every minimal basis. An empty MustSeed does not mean an absence of support: the root may have several alternative disjoint bases.

These sets do not measure truth, importance, or causal necessity outside the support contract.

6.3. Minimum size

If $B_{min}(q) \neq \emptyset$, define

$$\beta(q) = \min_{B \in B_{min}(q)} |B|.$$

For an empty family, $\beta(q)$ is undefined. Otherwise, it is the minimum cardinality of a seed basis in this record, not the number of physical causes, a degree of confidence, or a quantity of meaning.

7. Exact blocking of the root

Let $D \subseteq S$ be seeds disabled before the least closure is recomputed. The root is absent after blocking exactly when no minimal basis survives whole.

Theorem SG-HIT

$$q \notin Cl_R(S \setminus D) \Leftrightarrow \forall B \in B_{min}(q): B \cap D \neq \emptyset.$$

Proof

If some $B \in B_{min}(q)$ does not intersect D , then $B \subseteq S \setminus D$, and monotonicity gives the root after blocking.

Conversely, if the root remains after blocking, SG-LFP-3 gives a finite support tree with leaf set $L \subseteq S \setminus D$. The finite L contains an inclusion-minimal sub-basis $B \subseteq L$. Then $B \in B_{min}(q)$ and $B \cap D = \emptyset$, contradicting the hitting condition. $\square$

A minimum hitting set of $B_{min}(q)$ is a seed-blocking certificate. It is not a physical intervention or evidence of external causality.

8. Cycles in an AND/OR hypergraph

8.1. Dependency graph

The rule-dependency graph G_R has vertex set N and an edge

$$p \rightarrow h$$

for every $r \in R$, $p \in Prem(r)$, and $h = Head(r)$.

A cycle in G_R records a circular rule reference. It is not a seed.

Corollary SG-NOSEED

If no finite support tree for root q has leaves in S , then

$$q \notin \mu T_S.$$

In particular:

- $q \leftarrow q$ without seed q does not add q ;
- $a \leftarrow b$, $b \leftarrow a$ without seeds adds neither a nor b ;
- $b \leftarrow \{a, c\}$, $a \leftarrow b$ with seed c alone adds neither a nor b ;
- repeating a rule any number of times does not replace a missing seed leaf.

8.2. Grounded cycle

If a is a seed and the rules contain $a \rightarrow b$ and $b \rightarrow a$, the least closure contains both codes. A support tree for b ends at seed a ; the reverse rule is not required for its finite support.

For the AND rule $b \leftarrow \{a, c\}$, both branches are required. Seed a together with cycle $a \leftrightarrow b$ does not create the missing seed c .

8.3. OR alternatives

Suppose the rules are

$$\{s_1\} \rightarrow q, \{s_2, s_3\} \rightarrow q.$$

The root then has minimal bases

$$\{s_1\}, \{s_2, s_3\}.$$

Selecting one proof tree does not remove the existence of the other. Provenance that publishes only a randomly selected branch as the sole possible ancestry is incomplete relative to a family-of-supports audit.

9. CP-LFP-3 as a unary special case

Let V be a finite set of graph nodes, let $OIntro(s)$ be the set of protected introductions at node s , and fix an origin-tagged atom o . For every $s \in V$ with $o \in OIntro(s)$, create a seed code $b_{s,o}$ and a rule

$$Prem = \{b_{s,o}\}, Head = (s, o).$$

For every graph edge $u \rightarrow v$, create a unary claim rule

$$\{(u, o)\} \rightarrow (v, o),$$

that is, $Prem = \{(u, o)\}$ and $Head = (v, o)$. SG-LFP-3 then gives:

$$(v, o) \in \mu T_S \Leftrightarrow \exists s \in V (o \in OIntro(s) \wedge \text{существует finite support chain от } b_{s,o} \text{ к } (v, o))$$

The exact formula strings *существует ... от* and *к* have the secondary English reading *there is ... from and to*.

This is equivalent to the CP-LFP-3 path formula. An AND/OR hypergraph is a conservative extension of unary provenance, not a replacement for origin tags.

10. Binding, event, cause, application, purpose, and meaning

For each protected sort, one may construct a separate code universe and rule family, or use a single typed N . In both cases SG-LFP-3 establishes only rule support.

Examples of admissible record forms are:

$$\begin{aligned} & \{ \text{formal} : m, \text{binding} : b \} \rightarrow \text{application} : a ; \\ & \{ \text{sensor} - \text{record} : s, \text{criterion} : c \} \rightarrow \text{event} - \text{code} : e ; \\ & \{ \text{model} : M, \text{intervention} - \text{code} : i \} \rightarrow \text{cause} - \text{code} : k ; \\ & \{ \text{goal} : g, \text{application} : a \} \rightarrow \text{purpose} - \text{evaluation} : p . \end{aligned}$$

If those rules and seeds are registered, the least closure may contain the head code. It does not follow:

- that `binding:b` links a record to a referent;
- that an event occurred;
- that the model is a physical cause;
- that an application took place;
- that a goal was accepted or justified by anyone;
- that a purpose code is meaning;
- that a meaning code has an external referent.

SG-LFP-3 prohibits only provenance of a head code without finite seed support inside the declared rule record. A transition from a code to what it names remains a separate Phase 1–2 target/bridge contract.

11. Self-reference

A record may contain:

```
q: "q истинно"
rule: q <- q
```

The exact Russian string `q истинно` has the secondary English gloss `q is true`.

If `q` is not a protected seed and there are no other rules into `q`, the least closure does not contain `q`. Quotation marks, a quotation id, and a self-loop are not seeds.

If the contract adds a nullary constructor `ASSERT-q`, it is normalized into the protected base `RULE-SEED:ASSERT-q` and an inference to `q` with `premise_claim_ids=[]` and `required_base_ids=["RULE-SEED:ASSERT-q"]`. Then `q` belongs to the closure by that explicit reference. Within the contract, the support is the recorded base constructor, not self-confirmation by the payload. The status remains a formal code status.

The mutually referring lines

```
a: "b подтверждает a"
b: "a подтверждает b"
```

retain the exact Russian strings *b* подтверждает *a* and *a* подтверждает *b*. Their secondary English glosses are *b* confirms *a* and *a* confirms *b*.

They do not create a support tree without a seed leaf. They may belong to a non-least fixed point; this is examined in 03_FIXED_POINT_SELECTION_RU.md.

12. Admission gates

SG-LFP-3 applies only when:

1. N, S, R are finite and $S \subseteq N$; the executable projection uses namespace-tagged $B \sqcup C$;
2. every `base_id`, `claim_id`, and `inference_id` is unique; every `conclusion_id`, `premise_claim_ids`, and `required_base_ids` resolves in its own sort table;
3. after normalization, every abstract ordinary rule has non-empty $Prem(r)$; the executable checker requires the union of `premise_claim_ids` and `required_base_ids` to be non-empty for every inference;
4. every nullary constructor $\emptyset \rightarrow h$ has already been normalized into a fresh protected base b_r and ordinary rule $\{b_r\} \rightarrow h$; the executable inference to the conclusion explicitly references that `base_id` through `required_base_ids`;
5. the rules are positive: premises are checked by membership alone, without negation or deletion;
6. the operator is exactly (SG) and monotone;
7. closure is computed from $\emptyset$;
8. the support tree is finite and every internal node corresponds exactly to a rule record;
9. leaves are protected seeds, not payload strings;
10. no scheduler, initial state, oracle, or policy change is hidden from the base/readset.

The following constructions are outside SG-LFP-3 without a new contract:

- negation as failure;
- a rule q that depends on the absence of q ;
- priorities, weights, and defeat without an exact policy;
- deletion of an already added code;
- infinite proof trees;
- an unrecorded external oracle;
- probabilistic rule firing without the seed and schedule in the readset.

OUTSIDE-SG-LFP-3-NONMONOTONE applies to non-monotone constructions; the other listed violations receive no theorem status before a separate contract is given. This does not mean that another system is false.

13. Normative statuses

In this table, `VALID-SUPPORT-GROUNDING-LFP` and `INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT` are top-level schema statuses; `NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE` is an element of `reason_codes`; the remaining lines are protocol-level labels of theorems and controls, not members of the checker's `status` enum.

These are source protocol strings. Their appearance in the EN companion does not execute the source checker or inherit its verdict.

Status	Condition	Does not establish
<code>VALID-SUPPORT-GROUNDING-LFP</code>	each <code>declared_minimal_supports</code> exactly equals the computed inclusion-minimal antichain for the corresponding claim; grounded membership therefore equals μT_S	truth of a code
<code>FINITE-SUPPORT-TREE-FOUND</code>	the root has an exact finite tree with seed leaves	an external referent of the root
<code>NO-SEED-NO-ROOT-IN-LFP</code>	no support tree exists and the root is absent from least closure	absence of what is named outside the record
<code>MINIMAL-ROOT-BASES-ENUMERATED</code>	every inclusion-minimal basis is enumerated	physical causes of the root
<code>ROOT-SEED-CUT-VALID</code>	a disabling set intersects every minimal basis	an external intervention
<code>GROUNDING-CYCLE-PROPAGATES</code>	a cycle receives the head through a finite seed tree	that the cycle creates a seed
<code>PAYLOAD-SUPPORT-SPOOF-REJECTED</code>	text is not admitted as a seed/rule	falsity of the text
status <code>INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT</code> ; <code>reason_codes</code> contains <code>NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE</code>	an inference has both empty <code>premise_claim_ids</code> and empty <code>required_base_ids</code>	a rejection of nullary systems in general
<code>OUTSIDE-SG-LFP-3-NONMONOTONE</code>	the operator is non-monotone	a verdict about another operator

14. Required controls

1. **One seed.** $S = \{s\}$, root s : a one-node tree.
2. **AND.** Rule $\{a, b\} \rightarrow q$: a alone does not give the root; both together do.
3. **OR.** Rules $\{a\} \rightarrow q, \{b\} \rightarrow q$: each seed alone forms a basis.
4. **Self-loop without a seed.** $\{q\} \rightarrow q$: the root is absent.

5. **Mutual cycle without a seed.** $a \leftarrow b \leftarrow a$: both are absent.
6. **Grounded cycle.** Seed a plus mutual rules: both belong to the closure.
7. **Missing AND premise in a cycle.** Seed c , rules $\{a, c\} \rightarrow b, \{b\} \rightarrow a$: a, b are absent.
8. **Three alternative bases.** The checker enumerates every basis, not only the first proof found.
9. **Empty MustSeed.** Two disjoint bases have an empty intersection, while the root remains supported.
10. **Seed hitting set.** Partial disabling leaves a surviving basis; an exact transversal removes the root.
11. **Nullary normalization.** An inference with empty `premise_claim_ids` and `required_base_ids` is rejected; the normalized control creates a base and gives its exact `base_id` in the `required_base_ids` of the inference to the conclusion.
12. **Payload spoof.** A serialized rule in payload does not change the closure.
13. **Unknown id.** A rule with an unknown premise is rejected before closure.
14. **Base identity.** Repetition of one `base_id` is structurally rejected; two distinct `base_id` values with identical payload remain two base records.

15. Exact falsifiers

SG-SUPPORT-FALSIFIER

With the admission gates preserved, one of the following is supplied:

1. $q \in \mu T_S$, but no finite support tree with leaves in S exists;
2. a finite exact support tree for q exists, but $q \notin \mu T_S$.

SG-BASIS-FALSIFIER

A set B is supplied that:

- is included in the published $B_{min}(q)$, but does not derive the root or has a proper sub-basis; or
- is an inclusion-minimal root basis, but is absent from the minimal support-tree leaf sets.

SG-HIT-FALSIFIER

After seed deletion without changing the rules:

- the disabling set intersects every minimal basis, but the root remains; or
- a minimal basis survives whole, but the root disappears from the least closure.

What is not a falsifier

- an empty-premise rule without normalization;
- an infinite circular derivation;
- a non-monotone rule;

- a different fixed-point policy;
- a hidden oracle or scheduler;
- a claim that a code has or lacks an external referent.

These cases change the contract or lie outside its domain.

16. Maximal formulation

The exact Russian maximal formulation is:

В конечной положительной AND/OR-записи после нормализации nullary constructors, где каждое ordinary rule имеет непустое множество premises, root входит в наименьшее замыкание ровно тогда, когда существует конечное support tree, каждая ветвь которого заканчивается защищённым seed-введением. Все минимальные способы поддержать root совпадают с inclusion-minimal множествами seed-листьев; чтобы удалить root отключением seeds, необходимо и достаточно пересечь каждый такой basis. Циклические rule-ссылки могут участвовать в записи, но без конечной seed-ветви они не вносят root в least closure.

Secondary English gloss:

In a finite positive AND/OR record, after nullary constructors have been normalized and every ordinary rule has a non-empty premise set, the root belongs to the least closure exactly when a finite support tree exists whose every branch ends at a protected seed introduction. All minimal ways to support the root coincide with the inclusion-minimal sets of seed leaves; to remove the root by disabling seeds, it is necessary and sufficient to intersect every such basis. Cyclic rule references may participate in the record, but without a finite seed branch they do not place the root in the least closure.

The gloss is a separate written surface and does not replace the source line.

This line concerns code membership. It does not claim the presence or absence of what is named, the truth of the rules, physical causality, application, a goal, or meaning.

FP-SEL-3 (FPS-3): fixed-point selection and circular residue

0. Normative status

FP-SEL-3 is the normative identifier. FPS-3 is the short alias: **Fixed-Point Selection**.

CP-LFP-3 and SG-LFP-3 use the least fixed point. Yet the same positive cyclic equation may have multiple fixed points. This file separates three questions exactly:

1. which codes belong to the least fixed point;
2. which additional codes may be supported only by a circular fixed-point assignment;
3. which record selects one fixed point from among several.

The words least, greatest, stable, supported, true, and selected are names of formal policies/statuses. They do not introduce truth, a referent, an event, a cause, an application, a goal, or meaning.

Global status fence for the entire file: there is no separate executable FP-SEL verdict schema here. Every label in this file—including LEAST-FIXED-POINT-FROM-BOTTOM, GREATEST-FIXED-POINT-FROM-TOP, UNIQUE-SUPPORT-GROUNDED-FIXED-POINT, UNSUPPORTED-NONLEAST-FIXED-POINT, CIRCULAR-RESIDUE-LOCALIZED, FIXED-POINT-POLICY-SUPPLIED, POLICY-SENSITIVE-CODE, PAYLOAD-POLICY-SPOOF-REJECTED, OUTSIDE-FP-SEL-3-NONMONOTONE, and READSET-INCOMPLETE—is a protocol-level label, not an enum member of the top-level `status` field of the two executable CP/SG verdict schemas referenced by this file. The exact enums of those two schemas contain only:

- cyclic provenance: VALID-CYCLIC-PROVENANCE-LFP, INVALID-CYCLIC-PROVENANCE, INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT, CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS, CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL;
- support grounding: VALID-SUPPORT-GROUNDING-LFP, INVALID-SUPPORT-GROUNDING, INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT, SUPPORT-GROUNDING-INVARIANT-AUDIT-PASS, SUPPORT-GROUNDING-INVARIANT-AUDIT-FAIL.

Their separate `semantic_status` field remains NOT-MACHINE-VERIFIED. In particular, READSET-INCOMPLETE below is a protocol diagnosis of an incomplete projection, not an invented runtime status.

1. Base positive contract

The support contract SG-LFP-3 is used:

- a finite abstract code universe N ;
- protected seeds $S \subseteq N$;
- a finite set of positive rules R ;
- every abstract ordinary rule has a nonempty $Prem(r)$;

- executable bases/claims are embedded into N with distinct namespace tags; nullary constructors are normalized in advance into protected bases explicitly listed in the `required_base_ids` of the inference leading to the conclusion;
- the operator

$$T_S(X) = S \cup \{ \text{Head}(r) : r \in R, \text{Prem}(r) \subseteq X \}.$$

The operator is monotone on the finite lattice 2^N .

Its set of fixed points is

$$\text{Fix}(T_S) = \{ X \subseteq N : T_S(X) = X \}.$$

The least and greatest fixed points are denoted by

$$\mu T_S, \nu T_S.$$

They exist in the finite powerset domain. μT_S is computed by iteration from $\emptyset$; νT_S , by downward iteration from N :

$$L^0 = \emptyset, L^{n+1} = T_S(L^n);$$

$$U^0 = N, U^{n+1} = T_S(U^n).$$

For monotone T_S , the first sequence is increasing, the second is decreasing, and both stabilize.

least and greatest refer to the subset order on code sets. They do not mean “least true” and “most true.”

2. The interval of fixed points

Theorem FP-INTERVAL

For every $X \in \text{Fix}(T_S)$:

$$\mu T_S \subseteq X \subseteq \nu T_S.$$

Moreover,

$$\mu T_S = \bigcap_{X \in \text{Fix}(T_S)} X,$$

$$\nu T_S = \bigcup_{X \in \text{Fix}(T_S)} X.$$

Proof

By the definition of the least fixed point, it is included in every fixed point, while the greatest fixed point includes every fixed point. Because μT_S itself belongs to $\text{Fix}(T_S)$, the intersection of all fixed points cannot contain an element outside μT_S , and it also contains μT_S . The intersection therefore equals μT_S . The dual argument gives the union and νT_S . $\square$

Policy-sensitive region

Define

$$\Delta(T_S) = \nu T_S \setminus \mu T_S.$$

Then:

- a code in μT_S belongs to every fixed point;
- a code outside νT_S belongs to no fixed point;
- every code in $\Delta(T_S)$ belongs to the greatest fixed point and is absent from the least fixed point, so its membership depends on the fixed-point policy at least between these two admissible choices.

Policy-sensitive means a difference in code membership between formal selections. It does not mean indeterminacy of the world.

3. Support-grounded fixed point

3.1. Definition

A fixed point $X \in \text{Fix}(T_S)$ is called support-grounded if, for every $q \in X$, there is a finite SG-LFP-3 support tree with root q , exact rule IDs at the internal nodes, and protected seed leaves from S .

This definition requires a separate finite tree for every membership. The equality $X = T_S(X)$ alone is insufficient: it permits circular support.

Theorem FP-SEL-3: uniqueness of the support-grounded fixed point

$$X \in \text{Fix}(T_S) \wedge X \text{ support-grounded} \Leftrightarrow X = \mu T_S.$$

Proof

Let X be a support-grounded fixed point. For every $q \in X$, there is a finite support tree. By SG-LFP-3:

$$q \in \mu T_S.$$

Consequently, $X \subseteq \mu T_S$. Because X is a fixed point, leastness gives the reverse inclusion $\mu T_S \subseteq X$. Hence $X = \mu T_S$.

Conversely, SG-LFP-3 gives a finite support tree for every $q \in \mu T_S$, and μT_S is a fixed point. Hence it is support-grounded. $\square$

Thus, the least rule does more than select the smallest set: it selects the unique fixed point in which every accepted code has finite seed support under the declared rules.

This is a conditional characterization within the support contract. It does not make the seeds external grounds or make the root true.

4. Unsupported circular residue

Let $X \in \text{Fix}(T_S)$. Its nonleast residue is

$$\text{Res}(X) = X \setminus \mu T_S.$$

Theorem FP-RESIDUE

If $\text{Res}(X) \neq \emptyset$, then:

1. no $q \in \text{Res}(X)$ has a finite support tree with seed leaves;
2. for every $q_0 \in \text{Res}(X)$, there is a rule r_0 with head q_0 , premises in X , and at least one premise $q_1 \in \text{Res}(X)$;
3. by repeatedly selecting such a residue premise, in finite N one obtains a directed dependency cycle lying entirely in $\text{Res}(X)$, and a directed path from that cycle to q_0 .

Proof

Item 1 follows from SG-LFP-3: a finite support tree would place q in μT_S .

For item 2, $q_0 \notin S$, since otherwise $q_0 \in \mu T_S$. Because $X = T_S(X)$, there is a rule r_0 with head q_0 and $\text{Prem}(r_0) \subseteq X$. If all premises lay in μT_S , the closure property of the least fixed point would add q_0 to μT_S . Thus, at least one premise q_1 lies in the residue.

The argument repeats for $q_1, q_2, \dots$. All selected codes belong to the finite set $\text{Res}(X)$, so some code repeats. At every step there is a dependency edge

$$q_{i+1} \rightarrow q_i.$$

The repeated segment forms a directed cycle. The preceding finite chain gives a path from the cycle to q_0 . $\square$

Exact interpretation of the residue

Unsupported means that there is no finite tree all of whose branches terminate in protected seeds. It does not necessarily mean a complete absence of seeds from the residue rules.

For example:

$$S = \{s\}, \{q, s\} \rightarrow q.$$

The sets $\{s\}$ and $\{s, q\}$ are fixed points. The second fixed point uses the seed s , but the membership of q still requires q itself and has no finite support tree. The residue $\{q\}$ is circular even though the rule also reads a seed.

The exact formulation is therefore unsupported circular residue, not the unconditional line *никаких seeds нигде нет* (there are no seeds anywhere).

5. Minimal examples

5.1. Self-loop

$$N = \{q\}, S = \emptyset, R = \{\{q\} \rightarrow q\}.$$

Then

$$\mu T_S = \emptyset, \nu T_S = \{q\}.$$

Both sets are fixed points. *least* rejects seed-ungrounded membership; *greatest* accepts circular membership. The line q did not become true in either case: the formal policy-status changed.

5.2. Mutual cycle

$$a \leftarrow b, b \leftarrow a, S = \emptyset.$$

The least fixed point is empty; $\{a, b\}$ is a nonleast fixed point. Each code is supported by the other, but there is no finite seed tree.

5.3. Grounded cycle

If $a \in S$, the least fixed point contains a, b . Here the support tree for b has seed leaf a . The cycle carries an introduced code and is not its seed.

5.4. Multiple alternatives

If the root has one finite seed rule and one circular rule, the root already belongs to the least fixed point through the seed rule. The presence of an additional circular branch does not make the membership policy-sensitive. The checker must look for the existence of at least one finite support tree, not prohibit every mention of a cycle.

6. Fixed-point selection as input

6.1. Policy record

The fixed-point policy χ must specify:

- policy_id;
- operator_id;
- initial state;
- iteration direction (bottom-up, top-down, or another direction);
- synchronous/asynchronous schedule;
- rule priority and tie-breaker, if used;
- stopping criterion;
- random seed or a complete randomized-kernel contract;
- result serialization order.

For the standard least policy:

```
policy_id: LEAST-FROM-BOTTOM
initial_state: EMPTY
update: MONOTONE-SYNCHRONOUS
```

For the greatest policy:

```
policy_id: GREATEST-FROM-TOP
initial_state: FULL-CODE-UNIVERSE
update: MONOTONE-SYNCHRONOUS
```

The policy name does not replace the algorithm fields. The checker recomputes the selection.

6.2. Selection function

Let

$$Sel(N, S, R, \chi) \in Fix(T_S)$$

be a total selection procedure for accepted policy records.

The complete base projection ρ must include χ , or fix it unambiguously through an exact contract hash. If two executions have the same N, S, R but different policies and outputs, the policy is a distinguishing input.

Theorem FP-POLICY

Suppose two admissible indices k_0, k_1 have the same policy-free core

$$\rho_0(k) = (N, S, R),$$

but

$$\chi(k_0) \neq \chi(k_1), \text{Sel}(N, S, R, \chi(k_0)) \neq \text{Sel}(N, S, R, \chi(k_1)).$$

Then the selected fixed point is not a function of ρ_0 alone. It factors only through the extended input

$$\rho^x(k) = (\rho_0(k), \chi(k)).$$

Proof

A function has only one output for a given value of ρ_0 . Two required distinct outputs for the same argument are impossible. Once χ is added, the arguments differ. This is RBI-1 for fixed-point selection. $\square$

A globally fixed policy is also part of the formal contract, even if its line is not repeated in every instance. It does not arise from the cycle; it specifies how to read the cycle.

6.3. A policy capable of selecting a nonleast fixed point

A greatest, coinductive, stable, or other policy is not declared “false” merely because its description differs from `least`. Let the actually selected set be

$$X_\chi = \text{Sel}(N, S, R, \chi).$$

The protocol label `FIXED-POINT-POLICY-SUPPLIED` marks the presence of a complete χ . The label `UNSUPPORTED-NONLEAST-FIXED-POINT` is admissible only when

$$X_\chi \setminus \mu T_S \neq \emptyset,$$

that is, for an actually selected $X_\chi \neq \mu T_S$. The name `greatest`, `coinductive`, or `stable` alone is insufficient: if $\nu T_S = \mu T_S$, the `greatest` policy selects the same least fixed point and the `unsupported` label does not apply.

`FIXED-POINT-POLICY-SUPPLIED`

`UNSUPPORTED-NONLEAST-FIXED-POINT` # only if selected $X_\chi \setminus \mu(T_S)$ is nonempty

A contract that actually selects a nonleast residue may deliberately count circular support as sufficient. But it may not simultaneously declare `FINITE-SUPPORT-TREE-FOUND` for a residue that has no support tree.

7. Target selection after the fixed point

Let the target coordinate $Q: K \rightarrow Y$ be fixed independently, while the evaluator reads

$$V(k) = (N, S, R, \chi, \text{Sel}(N, S, R, \chi)).$$

If $V(k_0) = V(k_1)$ but $Q(k_0) \neq Q(k_1)$, exact export is impossible. For a relation-valued target with acceptable sets $A(k) \neq \emptyset$, zero-error export exists on a view fiber exactly when the full intersection is nonempty:

$$\bigcap_{k: V(k)=v} A(k) \neq \emptyset.$$

A fixed-point policy can distinguish cases only if it actually differs and is part of the readset. The name least, greatest, or stable does not select an external referent.

This applies separately to the following code coordinates:

- referent assignment;
- event classification;
- causal-model answer;
- application binding;
- goal/purpose choice;
- meaning/justification relation.

Even a complete fixed point is a record view. If the target differs while the view remains the same, recirculating the same view does not add a channel.

8. Self-reference and quotation

8.1. Self-assertion

Let the payload of node q be this exact string:

"q истинно"

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

"q is true"

Let the rule record contain $\{q\} \rightarrow q$. Without a protected seed, the least fixed point does not contain q . The greatest fixed point may contain q , but the source of the membership is the *greatest* policy, not the referent of the string истинно (*true*).

8.2. Mutual confirmation

Two codes may refer to one another and form a nonleast fixed point. FP-RESIDUE localizes the circular portion. The number of repetitions, the payload length, and the presence of quotation marks do not create a finite seed tree.

8.3. Self-interpreter

An interpreter that reads its own source code is added as nodes/rules/readset. If its execution starts from an exact seed state and least closure, every output code has a finite support tree. If the execution depends on a clock, scheduler, memory image, input stream, or oracle, those records belong in ρ . Reading its own text does not create an unread target.

8.4. This file

The line `FP-SEL-3` inside this carrier does not select a reading policy for its reader. The formal contract selected here specifies one read-as relation; it does not establish its own external referent, priority, or world truth.

9. Feedback, controller, and rocket

A feedback graph may contain the cycle

```
controller -> actuator-code -> sensor-record -> controller
```

CP-LFP-3/SG-LFP-3 do not state that physical feedback is absent. They require the record introductions to be enumerated:

- formal rule/state;
- symbol-to-signal binding;
- actuator interface;
- sensor transcript;
- event criterion;
- goal/setpoint;
- fixed-point/update policy.

If the sensor record enters as a seed, the cycle may carry it. If the goal code enters as a seed, the controller can compute consequences relative to it. If they are absent, a self-loop does not create the corresponding protected code in the least closure.

Even if the line `rocket flew` is conceded as an accepted event code, no purpose code follows from it. Two exact completion records may have identical formal, controller, sensor, and event views but different disjoint purpose acceptable sets. The purpose selector is then absent until a goal/purpose channel is supplied. Flight code and goal code are distinct coordinates.

Physical coupling, when admitted, is an added implementation channel. The edge name `actuator` is not motion, and a provenance edge is not a physical cause.

10. Admission gates

FP-SEL-3 admits an instance only if:

1. the powerset domain is finite;
2. T_S is specified exactly and is monotone;
3. for every inference, the union of `premise_claim_ids` and `required_base_ids` is nonempty; nullary constructors are normalized into bases with an explicit `required_base_ids` reference;
4. least is computed from $\emptyset$, and greatest from exact N ;
5. fixed-point equality is recomputed;
6. support trees are checked under SG-LFP-3;
7. the policy record is complete and belongs to the readset;
8. the scheduler, initial state, tie-breaker, randomness, and oracle output are not hidden;
9. a payload does not create a seed, rule, or policy;
10. target/acceptable sets are specified before the output audit.

Nonmonotone recursion may have no fixed point, may oscillate, or may depend on evaluation order. It receives

OUTSIDE-FP-SEL-3-NONMONOTONE

until another exact contract is supplied. This is not a negative verdict on a nonmonotone system.

An infinite graph, infinite regress, and transfinite iteration likewise fall outside the finite checker. Textual general fixed-point theory does not transfer machine status to an unsupplied infinite carrier.

11. Normative statuses

The status fence in §0 governs the entire table: every row below is a protocol-level label, not a schema enum or a machine verdict. READSET-INCOMPLETE in §12 has the same level.

Status	Exact condition	Does not establish
LEAST-FIXED-POINT-FROM-BOTTOM	bottom iteration stabilized at μT_S	external truth of membership
GREATEST-FIXED-POINT-FROM-TOP	top iteration stabilized at νT_S	superiority of the policy
UNIQUE-SUPPORT-GROUNDED-FIXED-POINT	FP-SEL-3 is satisfied	an external ground
UNSUPPORTED-NONLEAST-FIXED-POINT	the actually selected $X_x \setminus \mu T_S \neq \emptyset$	falsity of the nonleast contract
CIRCULAR-RESIDUE-LOCALIZED	a residue dependency cycle is supplied	a physical cycle

Status	Exact condition	Does not establish
FIXED-POINT-POLICY-SUPPLIED	a complete χ belongs to the base/readset	that the policy was selected by the world
POLICY-SENSITIVE-CODE	the code belongs to $\nu T_S \setminus \mu T_S$	semantic ambiguity
PAYLOAD-POLICY-SPOOF-REJECTED	a policy name in a payload did not change the contract policy	falsity of the payload
OUTSIDE-FP-SEL-3-NONMONOTONE	T_S is nonmonotone	a verdict on another semantics

12. Required controls

1. **Unique acyclic fixed point.** Least = greatest; the policy-sensitive region is empty.
2. **Empty self-loop.** Least is empty; greatest contains the loop code.
3. **Mutual seedless claim cycle.** Least is empty; the nonleast set is localized as residue.
4. **Grounded cycle.** A seed makes the code a member of least; it is no longer policy-sensitive.
5. **Seed plus self-dependence.** Rule $q \leftarrow \{q, s\}$: nonleast q remains unsupported.
6. **Alternative finite proof.** A circular rule is present, but a separate seed rule introduces the root into least.
7. **Bottom replay.** Every order declared by the exact synchronous policy reproduces the least bytes.
8. **Top replay.** Greatest is reproduced from the exact full universe.
9. **Policy pair.** One core, least and greatest, different outputs; the difference is attributed to χ .
10. **Policy-invariant code.** A code in least is present under both policies.
11. **Code absent from greatest.** The code is absent under every fixed-point choice.
12. **Payload spoof.** The line policy: greatest in a payload does not change the contract policy.
13. **Hidden schedule.** Different outputs under an unaccounted-for scheduler receive READSET-INCOMPLETE, not theorem status.
14. **Target conflict after identical selection.** The fixed point is identical and disjoint targets differ: export is rejected.
15. **Target overlap.** A shared acceptable code is accepted if the full fiber intersection is nonempty.

13. Exact falsifiers

FP-GROUNDING-FALSIFIER

With the SG-LFP-3 admission gates preserved, one of the following is supplied:

1. a fixed point $X \neq \mu T_S$, yet every $q \in X$ has a finite support tree with seed leaves;

2. μT_S contains a code with no finite support tree.

FP-RESIDUE-FALSIFIER

There is a nonempty $X \setminus \mu T_S$, but for some residue code, backward selection along actually firing rules can neither continue to a residue premise nor reach a dependency cycle—under finite N , positive rules, and exact fixed-point equality.

FP-POLICY-FALSIFIER

Two cases have the same policy-free core and different required policy-selected outputs, yet a single total function of that core alone produces the correct distinct output for each case.

FP-INTERVAL-FALSIFIER

An exact fixed point is supplied that does not contain the least fixed point or is not contained in the greatest fixed point, for the same monotone operator.

What removes the premises without falsifying the theorems

- a nonmonotone rule;
- an incomplete code universe for greatest iteration;
- a hidden initial state;
- changing rules during iteration;
- an unrecorded schedule/oracle/random seed;
- an infinite carrier;
- a post hoc target;
- a claim that a nonleast code has an external referent.

14. Maximal formulation

After nullary constructors have been normalized, in a finite positive rule record whose ordinary rules have nonempty premises, the least fixed point is the unique fixed point in which every accepted code has a finite support tree to protected seeds. Every additional membership in a nonleast fixed point contains an unsupported circular residue: finite backward tracing through the rules reaches a dependency cycle that cannot be reduced to a finite seed tree. Selecting least, greatest, stable, or another fixed point is a recorded policy and a distinguishing input. Repetition, self-reference, and feedback may carry or circulate codes already introduced; they do not select an unread referent, event, cause, application, purpose, or meaning.

No claim is made that every system must use the least policy. The exact fork is:

- least is the unique finite-seed-grounded fixed point;

- an actually selected $X_{\chi} \neq \mu T_S$ is a policy-supplied fixed point with circular residue; a greatest policy for which $\nu T_S = \mu T_S$ does not fall under this item;
- an external target is a separate coordinate, not created by the name of a policy or code.

SELF-PROOF IS NOT A SOURCE

Self-Reference, Mutual Checking, Mathematical Reading, the Rocket, and the Question “What For?” (Exact Russian Coordinate: *зачем*)

Phase 3 • application 4 • technical version 3.0.0 • August 28, 2026
Kirill Shebetov • beforeword / research

A record may return to itself.

A return is not an introduction of status.

Abstract

This file applies CP-LFP-3, SG-LFP-3, and FP-SEL-3 to the strongest objections to the record boundary. It admits self-referential formulas, self-hosting, mutual checking, proof-carrying records, recursive semantics, feedback control, and the strongest successful technical application. It even grants the full concession: the rocket exists, the calculation is correct, the calculation was used, the controller worked, the trajectory matched the calculation, and the rocket is flying.

The concession does not conflate distinct record relations. A line about its own proof status, an edge of mutual trust, a proof-checker output, a telemetry code, an event code, a causal-link code, and a goal code are read under different declared relations; their inputs need not be disjoint. In least cyclic provenance, every protected atom either has an origin-tagged introduction and a finite path or is absent. For every $q \in \mu T_S$, there is a finite seed-grounded support tree. If a particular nonleast or coinductive fixed point is selected, accepted, or used for a verdict, the corresponding fixed-point policy belongs to the complete readset. The mere existence of multiple fixed points does not yet record a selection of any nonleast policy.

Self-proof is therefore not rejected as a formal construction. Mutual checking is not declared useless. Technical success is not denied. A more precise boundary is recorded: a closed record carries introduced statuses under its rules, but does not become an independent source of truth, referent, event, cause, goal, or meaning status merely because it returns to itself.

0. The lines are supplied first

я доказал себя
мы проверили друг друга
это математика
это истинно

ракета полетела
 расчёт сработал
 мы знаем зачем
 в этом есть смысл

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

*I proved myself we checked one another this is mathematics this is true the rocket flew the
 calculation worked we know why this has meaning*

These eight exact lines come first. Their grammatical form does not create a speaker, a group of checkers, a mathematical object, truth, a rocket, an event, a goal, or meaning.

Even the phrase *сначала это строки* (*first, these are lines*) is a line in the present file. Its technical status appears only within the declared reading contract. No sentence code receives an exemption from the boundary it applies to other sentence codes.

Additional records are admitted below:

- graph record;
- parser record;
- rule record;
- proof record;
- checker record;
- binding record;
- sensor record;
- event record;
- causal-model record;
- goal record;
- fixed-point-policy record.

Their presence may change the declared output. It does not turn the name of a record into the thing named.

1. Relations that must not be conflated

The application specifies distinct relations:

Relation	What it assigns within the contract	What it does not assign on its own
$Contains_R(x, a)$	code a belongs to record x under exact procedure R	truth, proof, referent, occurrence
$a \in Intro_\tau(v)$	node v protectedly introduces atom a of sort τ	the object named by the atom code
$(s, \tau, a) \in P_v^\mu$	an origin-tagged atom belongs to the least provenance of node v	an external fact or truth
$Checks_C(p, q)$	checker contract C accepts proof code p for claim code q	unconditional truth of q
$Refers_B(a, x)$	binding contract B accepts reference code (a, x)	presentation of x outside B
$Occurs_P(e)$	protocol P assigns occurrence status to event code e	a physical event outside record relation P
$Causes_M(x, e)$	model/procedure M accepts a causal-link code	causation outside M
$Endorses_N(g)$	normative contract N accepts goal code g	obligation or meaning

Without a bridge, the following transitions do not follow:

$$Contains \not\Rightarrow Intro \not\Rightarrow Checks \not\Rightarrow Refers \not\Rightarrow Occurs \not\Rightarrow Causes \not\Rightarrow Endorses.$$

The reverse transitions are not supplied either. Even when an occurrence is conceded as having taken place, it does not follow that the earlier record contained the event code or a causal model. Even when a formal proof is accepted, a reference binding does not follow.

2. The exact Phase 0 → Phase 1 → Phase 2 → Phase 3 line

The application rests on four successive constraints.

Phase 0

Type A was not derived from M -records alone under empty bridge dependencies. Every accepted application record retained bridge provenance.

Phase 1

The same complete input could not deterministically recover two conflicting independently fixed targets:

$$\rho(k_0) = \rho(k_1), Q(k_0) \neq Q(k_1) \Rightarrow Q \neq F \circ \rho$$

for every function F of the declared input.

Phase 2

If a conclusion in an immutable DAG correctly distinguished the target, a distinguishing origin-tagged atom occurred in the ancestry of the conclusion. A later node did not become the first source of an already fixed earlier verdict.

Phase 3

Ordinary ancestry is replaced by least cyclic provenance. Even when dependencies form a self-loop or SCC, an atom in the least fixed point has a protected introduction and a finite path. If one state is selected, accepted, or used for an output from among multiple fixed points, the selection policy becomes part of ρ . The multiplicity of $\text{Fix}(T_S)$ is not itself a record of a choice having been made.

The application line:

```
formal trace
+ bridge
+ origin
+ cyclic propagation
+ fixed-point policy
-> условный application status
```

Occurrence-local English reading of the final line:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

-> conditional application status

The deletion of any field from the record is not proved by a word that remains.

3. A single node declaring itself proved

Let the graph have one node v and a self-loop

$$v \rightarrow v.$$

The node payload says:

```
claim: v_is_proved
reason: v
```

At the same time:

$$OIntro(v) = \emptyset.$$

The provenance operator is

$$\Phi_G(X)_v = OIntro(v) \cup X_v = X_v.$$

Iteration from the empty state gives

$$P_v^{(0)} = \emptyset, P_v^{(n)} = \emptyset$$

for every n . Consequently:

$$P_v^\mu = \emptyset.$$

The payload v_is_proved is present under *Contains*. But, for every $a \in A_{FORMAL}$, the origin-tagged atom $(v, FORMAL, a) \notin P_v^\mu$. The *Contains* relation is not declared equal to membership in least provenance.

What can change the result

Least provenance P_v^μ may change if a new record input changes protected introductions or reachability from an introducer to v . In particular:

1. a local introduction of a proof atom;
2. an upstream seed;
3. a protected *Intro* constructor that separately introduces a code previously occurring only in a payload;
4. a change to the admissible graph that creates or removes an introducer-to- v path.

A nonleast/coinductive policy selects a different state but does not change the former P_v^μ . An oracle or hidden state removes *COMPLETE-READSET* until it is presented as a new input.

Each item is a new record input. None was present in the empty self-loop merely because it was named later.

4. A self-loop with an explicit seed

Now let v protectedly introduce

$$a \in Intro_{FORMAL}(v).$$

Then:

$$(v, FORMAL, a) \in P_v^\mu.$$

This is a positive control. Phase 3 does not prohibit a self-loop from receiving an atom. It identifies the exact origin: a local introduction at v .

Distinct questions nevertheless remain:

- whether a protected formal atom is present;
- whether the checker accepted a proof trace;
- whether the atom corresponds to an external claim;
- whether the claim has a truth relation assigned to it;
- whether the claim concerns anything outside the formal system.

CP-LFP-3 answers only the first question. $Checks_C$ may answer the second. The rest require other contracts.

5. Two nodes checking one another

Let:

$$u \rightarrow v, v \rightarrow u.$$

Payload:

```
u: v is valid
v: u is valid
```

Protected introductions are empty:

$$OIntro(u) = OIntro(v) = \emptyset.$$

The least provenance of both nodes is empty. The number of mutual references, identical signatures, matching hashes, and repetitions of the word `valid` are not seeds.

If, for some $\tau \in T_3$, node u receives a local protected introduction

$$t \in Intro_\tau(u),$$

the origin-tagged atom propagates:

$$(u, \tau, t) \in P_u^\mu \cap P_v^\mu.$$

The origin tag u is not erased. Mutual checking may:

- carry trusted-root status;
- compare outputs;
- detect discrepancies under a recorded criterion;
- increase redundancy;
- localize failure;
- build a certificate chain.

Mutuality alone does not create an independent root.

6. A ring of multiple checkers

The result does not depend on the cycle length. For

$$v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_{n-1} \rightarrow v_0$$

source-free least provenance remains empty with respect to an atom introduced nowhere.

If atom a is introduced at v_j , it reaches every node reachable from v_j . In a strongly connected ring, all nodes are reachable. But n copies do not turn one seed into n independent origins: the origin tag remains v_j .

The number of confirming payload lines measures the number of lines. It is not a count of independent grounds without a separately declared independence contract.

7. Support graph: a reference is not a ground

For a proof application, a dependency edge and a support tree are distinct.

The rule

$$\{p_1, \dots, p_m\} \Rightarrow q$$

may introduce q into the least closure only after all premises are present. Every $q \in \mu T_S$ has a finite support tree whose leaves are seeds; this statement does not cover an accepted nonleast/coinductive root.

Self-support

$$\{q\} \Rightarrow q$$

without a seed does not place q in μT_S .

Mutual support

$$\{p\} \Rightarrow q, \{q\} \Rightarrow p$$

without a seed places neither p nor q in μT_S .

Seeded mutual support

Given seed p :

$$p \in S \subseteq \mu T_S \Rightarrow q \in \mu T_S.$$

The support tree for q ends in leaf p . The cycle may be part of the derivation graph, but it does not replace the leaf.

Alternative bases

If the root is derived either from $\{a, b\}$ or from $\{c\}$, the minimal bases are

$$\{\{a, b\}, \{c\}\}.$$

Selecting one proof does not erase the second basis or make the root unconditional. Corollary SG-BASIS preserves the antichain of minimal bases; this is not itself the SG-LFP-3 equivalence.

An ordinary rule without premises is not disguised as a proof code without a base. If a nullary constructor is required, a separate protected base—for example, `RULE-SEED:<rule-id>`—is created before serialization, and the inference leading to the conclusion explicitly lists this `base_id` in `required_base_ids`; the checker does not normalize the input itself and rejects simultaneously empty claim/base premises with reason code `NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE`.

8. A proof checker checking its own code

Consider the records:

```
source-code
compiler-code
binary-code
proof-object
checker-code
execution-record
verdict-code
```

The strongest internal case may be admitted:

- the source describes the checker;
- the checker checks a proof object about the source;
- the binary runs the checker;
- the checker accepts the proof object;
- the output is `VALID`;
- the complete exact trace is published.

The accepted result is

$$Checks_C(p, q) = 1.$$

It is not subtracted. Phase 3 does not rename `VALID` as `INVALID`.

But the dependency chain may include:

- parser semantics;
- inference rules;

- compiler translation;
- runtime;
- hardware execution record;
- I/O binding;
- fixed-point policy;
- trusted axioms or seeds.

If the chain is cyclic, CP-LFP-3 describes the propagation of protected atoms through it. If the claim has a support tree, SG-LFP-3 identifies the seeds. Neither self-hosting nor feeding the output back to the input removes these records.

The line `checker проверил checker` (*the checker checked the checker*) does not assign independence to the check. Independence is a separate relation with a separate contract.

9. Self-proof and levels of claim

The word самодоказательство (*self-proof*) may conceal different targets:

Target code	Possible exact question
SYNTAX-WELL-FORMED	did the parser accept the bytes?
PROOF-TERM-ACCEPTED	did the kernel accept the proof term?
DERIVABLE-IN-C	is there a trace under rules $\mathcal{C}$?
CHECKER-EQUIVALENCE	do two checkers produce identical verdicts on the declared corpus?
IMPLEMENTATION-REFINES-SPEC	was a formal refinement proof accepted?
SELF-PROPERTY-CODE	did the system output the code of a declared property about its own record?
WORLD-TRUE	does the claim correspond to anything outside the formal contract?

The first six can be defined and checked within exact records. The seventh is not inherited from them without a bridge.

Even if the system outputs a formula code read as `эта система непротиворечива` (*this system is consistent*), the application of that result is limited by the exact formal semantics. The present work makes no assertion about the possibility of such a proof in every particular calculus. It records only the record-status boundary: an accepted formula code does not become an external guarantee through a reading alone.

10. Fixed-point policy as part of the proof contract

Let a self-loop satisfy the equation

$$X = X.$$

The equation admits multiple fixed points. Three contracts may record:

```
policy: least
policy: greatest
policy: declared-selector:S
```

With the same equations, they may select different states. Consequently, the policy belongs to the complete input whenever the contract selects, accepts, or uses one of these states for a subsequent verdict. The set $\text{Fix}(T_S)$ itself is determined by the operator and does not show that any selector has already been applied.

A nonleast state is not declared false. Coinductive semantics may be exact for its task. Nor must a greatest policy produce nonleast residue: $\nu T_S = \mu T_S$ is possible. Unsupported residue arises only for the actually considered $X \neq \mu T_S$. The following statement is incomplete:

атом присутствует, потому что equation замкнута

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

the atom is present because the equation is closed

The exact status is:

```
атом присутствует в fixed point,
выбранной policy P
```

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

the atom is present in the fixed point selected by policy P

If P changes, the record changes. If P is hidden, the declared readset is incomplete.

FP-SEL-3 supplies a diagnostic: membership outside μT_S has no finite seed-grounded support tree under the declared least-grounding contract. It may depend on some seeds and still retain a mandatory circular branch; the exact term is therefore unsupported circular residue, not residue без каких-либо seeds (*residue without any seeds*). For unary Φ_G , the separate analogue is the absence of a finite introduction-to-node path witness outside $\mu \Phi_G$.

11. Mathematics begins in this record as a reading contract

The exact carrier is not declared to be mathematics on its own. The access sequence is

carrier < decoder < tokenizer < parser < READ-AS-MATHEMATICAL < rules < proof trace.

Even линия (*line*), число (*number*), символ (*symbol*), формула (*formula*), and математика (*mathematics*) are classifying strings. A reading contract may correctly assign mathematical syntax to a carrier. This is constitutive status within the practice.

If the reading contract refers to its own output:

этот carrier математический,
потому что процедура, читающая его как математический,
выдала READ-AS-MATHEMATICAL

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

this carrier is mathematical because the procedure that reads it as mathematical returned READ-AS-MATHEMATICAL

Under this reading, the classification output depends on the decoder, parser, and classification rule. A subsequent formal status additionally depends on the inference rules and fixed-point policy. Self-reference does not place these inputs into the carrier before it is read.

The exact result does not require the historical line математика возникла из языка (*mathematics arose from language*). No historical-origin record is assigned. Only the order of records is supplied: within the presented contract, mathematical classification appears after the reading procedure and does not retroactively ground its own input.

12. Internal semantics is admitted in full

Formal records may contain:

$$Model(m), Sat(m, \varphi), True(\varphi), Denotes(a, x), Applies(r, q).$$

They may participate in recursive equations and have least, greatest, or stable semantics. Phase 3 does not remove them.

For a deterministic evaluator, these records are identical on an identical complete input, including the random seed, schedule, oracle output, and transcript if they are read. If the fixed-point policy belongs to the input, the result is conditional on it. If the intended model, external binding, or target differs while the internal records remain identical, another internal loop does not select the difference.

The line $True(\varphi)$ may be a correctly formed formula code of an internal theory. It does not acquire the unconditional relation WORLD-TRUE from the coincidence of the word True.

13. The strongest case for the rocket argument

The following points are admitted simultaneously:

1. the rocket exists;
2. the rocket was built;
3. the formal calculation was performed correctly;
4. the calculation was used in the design;
5. the software implements the controller under the accepted refinement contract;
6. the controller physically participated in guidance;
7. the sensors and telemetry worked;
8. the measurement record matched the prediction record under the accepted metric;
9. the rocket event received occurrence status;
10. the causal model accepted a link between the control chain and the flight event;
11. the rocket is flying.

These points are not derived from the characters in the file. They are supplied as the strongest concession to the objection. Thus, the phrase *но ракета действительно полетела* (*but the rocket really did fly*) is already included.

Let the following role codes be given:

m = formal calculation record ,
 b = application binding record ,
 i = implementation/refinement record ,
 c = controller record ,
 t = telemetry record ,
 e = event/occurrence record ,
 ℓ = causal-link record ,
 f = fit-criterion record ,
 g = goal record .

These are distinct role codes. Neither equality nor inequality of carriers follows from the differing labels: record identity and exact links are specified separately by IDs, edges, and rules.

14. The rocket feedback loop

The technical chain may be cyclic:

model → controller → actuation → sensor → telemetry → model.

A feedback loop can correct the trajectory. Cyclicity may be necessary for the system to work. Phase 3 does not deny this.

The following labels are atom-role codes, not new sorts. Each code separately receives one of the eight sorts in T_3 and enters through a protected record:

- formal atom—through a formal proof/checker;
- binding atom—through a mapping record;
- implementation atom—through a refinement/certificate record;
- measurement atom—through a sensor protocol;
- event atom—through an occurrence protocol;
- fit atom—through a criterion;
- causal-link atom—through a model/procedure;
- goal atom—through a goal input.

After introduction, the loop may carry and use the atoms. CP-LFP-3 prohibits only the hidden substitution of circulation for introduction.

If telemetry and model mutually calibrate one another, exact iterative records are admissible. But the initial calibration, metric, update rule, stopping rule, and accepted fixed-point policy are inputs. The word самокалибровка (*self-calibration*) does not remove them.

15. What exactly follows from a successful flight

The strongest concession allows an extended technical end-to-end record without a goal coordinate to be accepted:

$$r^+ = (m, b, i, c, t, e, \ell, f).$$

On it, the following may be checked:

- formal acceptance;
- implementation refinement;
- controller trace;
- measurement/prediction fit;
- occurrence status;
- causal-link status;
- performance criterion.

This is a strong result of the extended chain. It does not turn

$$m$$

into

$$(m, b, i, c, t, e, \ell, f).$$

The flight event may be a descendant input to a new verdict. It does not become the content of an earlier formula. Cyclic feedback after launch does not write itself backward into the bytes of an earlier calculation record.

The phrase *формула участвовала в создании ракеты* (*the formula participated in creating the rocket*) may be accepted under a causal/application contract. It does not entail *ракета находилась в формуле* (*the rocket was in the formula*). Participation in a chain and containment in a record are distinct relations.

16. The Question “What For?” After the Full Concession (Exact Russian Coordinate: *зачем*)

Even after every technical point has been accepted, different targets remain. The labels in the table are claim-role codes, not sorts; the sort of each code is selected separately only from T_3 :

Target	Exact code status	Additional requirement
cause	a causal link is accepted by model M	model, intervention/observation contract
design-intention	an intention code is linked to the design record	source and binding intention record
goal	a goal code is recorded	goal introduction
endorsement	the goal received an endorsement code	normative/agent contract
fit	the goal satisfies a criterion code	criterion and evaluation
justification	a justification code is accepted	separate rules/seeds
meaning	a meaning code is recorded	declared meaning contract

A calculation may optimize an already introduced utility:

$$x^* \in \arg \max_x U_g(x).$$

It does not select g from the formal core if two admissible continuations have the same core and different goal codes.

Consider the following cyclic repetition:

цель оправдана, потому что она наша цель
наша цель является целью, потому что она оправдана

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

the goal is justified because it is our goal our goal is a goal because it is justified

This repetition forms a mutual-support graph. In the absence of other bases and rules, both status codes are absent from μT_S . Given an accepted base, the derivation may proceed, while the `base_id` remains a leaf.

The exact boundary is stronger than the answers `смысл есть` (*there is meaning*) and `смысла нет` (*there is no meaning*). Both lines are answer codes. Neither is exported autonomously from the same formal/application input.

17. One hundred, two hundred, three hundred passes

Fix a seed set S , positive rules R , and the least policy. For two claim codes g, d , the only rules are $g \Rightarrow d$ and $d \Rightarrow g$, with $g, d \notin S$. Iteration of the support operator gives

$$C_S^{(n+1)} = T_S(C_S^{(n)}), C_S^{(0)} = \emptyset.$$

Then:

$$g, d \notin C_S^{(100)} \cup C_S^{(200)} \cup C_S^{(300)}$$

with S, R , and the least policy unchanged. Other seeded codes in these states need not be absent.

The tokens 100, 200, and 300 are not goal introductions. The duration of traversal does not create a new field.

If a protected goal base/introduction or a rule deriving a goal code is added at stage 301, a new result may become selectable. This is a new input, not the late discovery of a goal inside the former empty cycle.

No prohibition on every future channel follows. An undescribed future channel is not covered. For each fixed stage, the negative result is complete relative to its exact readset.

18. Constitutive convention

Objection:

In mathematics, a rule itself institutes a status. If a contract says that a fixed point is accepted, no external bridge is required.

The concession is admitted. A constitutive convention may assign formal status within a practice. If

policy: greatest

and the checker correctly computes the greatest fixed point, selected membership has status under that contract.

But the convention is a record input. It is not extracted from an empty self-loop. And the instituted formal status does not inherit occurrence, external reference, cause, or justification without a separate bridge.

Phase 3 does not dispute an internal rule of the game. It prohibits erasing that rule from the provenance after a result has been obtained.

19. Coinduction

A coinductive proof may legitimately use cycles and a greatest fixed point. This is not a falsifier of CP-LFP-3, because the fixed-point policy changes.

The exact statuses

LEAST-PROVENANCE-MEMBERSHIP
GREATEST-FIXED-POINT-MEMBERSHIP
STABLE-MODEL-MEMBERSHIP
DECLARED-POLICY-MEMBERSHIP

are not merged.

Coinductive status may be mathematically strong and useful. When it is applied to a stream, transition system, or bisimulation code, it remains conditional on the exact definitions and policy. The word *бесконечный* (*infinite*) does not become an infinite object outside a reading contract.

20. Mutual verification and independence

Two systems may:

- have different implementations;
- compare outputs;
- run on different devices;
- sign records with different keys;
- check one another's certificates.

Such a scheme can supply new distinctions because the readset contains different code, execution, key, and transcript records. They are inputs/channels; introduction occurs only through protected Intro.

If both systems depend on one specification, compiler, corpus, or trust root, mutuality does not make the dependencies independent. If independence is formalized and checked, its status concerns the declared relation.

The conclusion is therefore not взаимная проверка ничего не даёт (*mutual verification yields nothing*). The exact result is:

The output of mutual verification depends on the distinctions, rules, and channels represented in the complete readset; a ring of references without seeds does not itself introduce a protected root.

21. Self-modification and a changing graph

A self-modifying system may rewrite rules, introductions, or topology. The complete frozen instance then changes:

$$I_0 \neq I_1.$$

The equality $G_0 \neq G_1$ follows only when V or E changes; replacing introductions or support rules may change the instance while leaving the graph unchanged. CP-LFP-3 applies separately to each frozen instance or to an extended transition record. Changing protected Intro may introduce a CP atom in the new instance; a new support rule may derive an SG code. The rule itself is not a CP introduction.

The phrase система создала собственное правило (*the system created its own rule*) may receive status under a transition contract. It does not follow that the rule was absent as a mechanism/input or that the later record was contained in the earlier bytes.

Mutable history also does not establish physical time. Birth labels, timestamps, and hashes remain codes without a separate publication/attestation channel.

22. The strongest objections

22.1. “If a Proof Is Closed, It Is Complete” (Exact Russian Source Wording: Если proof замкнут, он завершён)

Closure of a dependency graph and groundedness of a support tree are different properties. A seedless dependency cycle is closed but has no seed leaves in the least closure.

22.2. “Every Participant Trusts the Next” (Exact Russian Source Wording: Каждый участник доверяет следующему)

This describes trust edges. If the trust rule is coinductive, the policy must be declared. If grounded trust is required, a protected seed/trust root is needed. The word *доверяет* (*trusts*) does not create trust outside the contract.

22.3. “The System Itself Selected the Least Fixed Point” (Exact Russian Source Wording: Система сама выбрала least fixed point)

If the selection is performed by an algorithm, the code and policy of that algorithm belong to the readset. If two policies produce different outputs for identical equations, the equations were not the complete input.

22.4. “A Proof Assistant Guarantees a Theorem” (Exact Russian Source Wording: Proof assistant гарантирует theorem)

Kernel acceptance may be admitted. The guarantee concerns the exact kernel/specification contract. Intended application, hardware adequacy, occurrence, and goal are not inherited automatically.

22.5. “The Rocket Proves the Formula” (Exact Russian Source Wording: Ракета доказывает формулу)

Flight may supply measurement/fit evidence for an application record. It does not turn an event into a syntactic derivation or assign proved status to every formula code. *Entails* and *Occurs* are distinct.

22.6. “Success Is What Gives It Meaning” (Exact Russian Source Wording: Успех и есть смысл)

This is a possible constitutive definition if *meaning* := performance-success is recorded as a contract. The meaning code is then computed relative to it. Acceptance of the definition itself is not derived from success.

22.7. “Language Has No First Source” (Exact Russian Source Wording: Никакого первого источника у языка нет)

Phase 3 does not assert a physical or historical first cause. Origin means a protected introduction in the exact graph. No historical-origin assignment for language or mathematics is recorded.

22.8. “People Understand Without Records” (Exact Russian Source Wording: Люди понимают без записей)

The line about people and understanding is not used as a premise. If an observation, behavioral, or neural record is added to the application, it remains a new record with a separate reading contract. A name does not present the named thing.

23. Exact application falsifiers

The application of CP-LFP-3 to self-proof is incorrect if:

1. a protected proof atom was in fact introduced but is described as unseeded;
2. the checker uses a nonleast policy hidden under the label `least`;
3. a payload is accepted as a protected introduction contrary to the contract;
4. the graph/rules change between iterations but are declared fixed;
5. a hidden oracle or external memory affects the verdict but is absent from ρ ;
6. nonmonotone negation/default rules are substituted into the positive-union theorem.

The application of SG-LFP-3 is incorrect if:

1. a seed leaf is omitted from the support tree;
2. a rule with an empty premise set is not normalized into a protected base and the inference leading to the conclusion does not include its `base_id` in `required_base_ids`;
3. a rule does not require all listed premises;
4. an alternative basis is removed to obtain the desired uniqueness;
5. a cycle has an incoming seed but is classified as source-free;
6. a root is accepted under a coinductive policy but called least-grounded.

The rocket application is described incorrectly if:

1. the strongest concession is covertly used as a derivation from the lines;
2. event, binding, cause, goal, and justification are conflated;
3. a telemetry record is presented as the event without an occurrence protocol;
4. a causal model is presented as unconditional causation;
5. technical success is presented as goal endorsement;
6. absence of a goal is presented as a proof of `смысла нет` (*there is no meaning*).

24. Self-application of this file

The following exact line appears in this file:

```
САМОДОКАЗАТЕЛЬСТВО НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ
```

Occurrence-local English reading:

The following editorial reading is separate from and remains outside the exact fenced source sequence:

```
SELF-PROOF IS NOT A SOURCE
```

This line is not a registered C-PUBLIC-TITLE. The current `beforeword.phase3-self-audit.v1` registers the title only from `00_MANUSCRIPT_RU.md`. The following block is therefore an unregistered protocol-level annotation, not an actual record in `self_claims_phase3.json`:

```

annotation_scope: 04_APPLICATION_SELF_PROOF_RU.md
registration_status: UNREGISTERED-PROTOCOL-ANNOTATION
proposed_kind: title
proposed_expected_status: UNBOUNDED-STANDALONE
semantic_status: NOT-MACHINE-VERIFIED

```

UNREGISTERED-PROTOCOL-ANNOTATION is not an enum member of the current self-claims schema. Machine registration of this title requires extending the schema, self-audit allowlist, and `self_claims_phase3.json`, and assigning a new `claim_id`, exact document/span, and hashes. Until such an extension is supplied, the current contract does not assign an `expected_status` field to the title of 04.

Three theorem claims are registered:

```

claim_id: C-THEOREM-CP-LFP-3
claim_id: C-THEOREM-SG-LFP-3
claim_id: C-THEOREM-FP-SEL-3
expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY
semantic_status: NOT-MACHINE-VERIFIED

```

The applications in this file are textual consequences without a separate registered status and remain at `semantic_status: NOT-MACHINE-VERIFIED`. A bounded fixture claim receives `expected_status: VERIFIED-IN-DECLARED-ARTIFACT`. The actual top-level `status` of a separate checker verdict must be one complete literal enum member of the corresponding verdict schema:

```

VALID-CYCLIC-PROVENANCE-LFP
INVALID-CYCLIC-PROVENANCE
INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT
CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS
CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL
VALID-SUPPORT-GROUNDING-LFP
INVALID-SUPPORT-GROUNDING
INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT
SUPPORT-GROUNDING-INVARIANT-AUDIT-PASS
SUPPORT-GROUNDING-INVARIANT-AUDIT-FAIL
semantic_status: NOT-MACHINE-VERIFIED

```

The ten status lines belong to two different verdict schemas and do not form a common wildcard enum. The first three values in each schema concern an instance verdict; the last two concern a contract-audit verdict. The `semantic_status` field must be retained with every one of them.

No status becomes any of the following through a single writing:

```

WORLD-TRUE
SOURCE-ATTESTED
REFERENT-PRESENT
EVENT-OCCURRED
GOAL-JUSTIFIED

```

If this file refers to one of its own sections as a ground, a dependency edge is formed. A grounded claim still requires a support tree with seeds. SHA-256 may fix byte identity within a cryptographic contract; it does not prove a creation date or physical history without a separate channel.

25. Boundary of the application claims

Recorded within the declared contracts

1. An unseeded self-loop does not introduce a protected atom into least cyclic provenance.
2. An unseeded mutual-verification cycle does not form a grounded proof root.
3. An explicit CP introduction retains its origin tag during propagation; an explicit SG seed remains a support leaf.
4. Self-checking and self-hosting do not remove the parser, rules, compiler, runtime, axioms, and policy from the complete contract.
5. Nonleast/coinductive membership depends on a separately declared fixed-point policy.
6. Mathematical status appears after a read-as contract and does not transfer the contract backward into the carrier.
7. A maximally successful rocket feedback loop propagates typed atoms introduced by the corresponding binding, sensor, event, fit, and causal-link records, but does not place the records themselves in the earlier formula.
8. Technical success does not select a goal, endorsement, or justification from an identical pre-goal input.
9. Repeating an unchanged unseeded least cycle does not create an atom at stages 100/200/300.
10. CP-AST preserves the no-export boundary for a conclusion computed from a base and least cyclic provenance.

Outside the recorded conclusion set

- that self-reference is prohibited;
- that mutual verification is useless;
- that a proof assistant is erroneous;
- that coinduction or a greatest fixed point is false;
- that formal theorems lack internal inferential force;
- that the rocket does not exist or does not fly;
- that the calculation did not participate in the flight;
- that technical success is absent;
- that a goal, justification, or meaning exists or is absent;
- that a reading contract exhausts any world;
- that checker status is external truth;
- that mathematical codes lack constitutive status;
- that mathematics historically arose from language;

- that no future channel will appear;
- any assessment assigning historical novelty or priority.

26. Conclusion

Self-reference may be an admissible form. Mutual checking may increase redundancy. A self-hosted checker may accept a proof about its own record. Recursive mathematics may have exact fixed-point semantics. A feedback controller may guide a rocket. The rocket may fly.

None of these concessions makes a closed line the first source of an absent status.

If least provenance contains an atom, there is an introduction and a path. If least support closure contains a root, there is a finite support tree with seed leaves. If a nonleast fixed point is selected as the accepted state, the selection policy belongs to the input; the mere existence of a fixed point does not require this. If a rocket verdict is accepted, the binding, implementation, sensor, event, criterion, and causal-model records belong to the extended chain. If a goal is supplied, it is a goal input. If a justification rule is supplied, it is a new contract.

The exact closing line therefore does not deny the result:

Self-proof may be an accepted formal record.

It is not its own missing ground.

And the question *зачем* (why) is not closed by repeating the output:

A calculation may carry a goal.

A cycle of calculations does not select why it should be accepted.

Source status

The application follows the internal line of Phases 0–3 and uses only the declared objects of those phases. External predecessors are not premises. No historical-priority assessment or global-novelty assessment is recorded here.

05. ADVERSARIAL REVIEW AND SELF-APPLICATION OF PHASE 3

A CYCLE IS NOT A SOURCE

0. Purpose and fixed boundary

This file attacks the strongest statements of Phase 3 before their public use. The purpose of the review is not to replace an exact negation with a softer formulation, but to separate the proved invariant from extensions that rely on an unwritten reading.

The words `цикл` (cycle), `источник` (source), `атом` (atom), `основание` (basis), `правило` (rule), `доказательство` (proof), `успех` (success), `ракета` (rocket), `полёт` (flight), `цель` (goal), and `смысл` (meaning) enter here as exact recorded forms. They do not present what they name. Type, origin, seed, edge, fixed-point policy, and status are protected fields of the declared protocol, not properties of anything outside it.

The unqualified title is the string `UNBOUNDED-STANDALONE`. Its exact content is given only by three results:

- `CP-LFP-3` (CFP-3) — Cyclic Provenance — Least Fixed Point;
- `SG-LFP-3` (GST-3) — Grounded Support Theorem;
- `FP-SEL-3` (FPS-3) — Fixed-Point Selection.

All three results concern finite positive record systems with frozen constructors: nullary constructors are prenormalized into protected bases, and ordinary rules have nonempty premises. They establish neither reference, event, physical causation, usefulness, purpose, justification, nor the absence of anything named.

Global status-layer fence

This fence applies throughout the file, including the tables in sections 10 and 11.

1. The `expected_status` field of self-claims uses only the full values `UNBOUNDED-STANDALONE`, `TEXTUAL-THEOREM-IN-DECLARED-METATHEORY`, `VERIFIED-IN-DECLARED-ARTIFACT`, and `NOT-ASSESSED` in the positions permitted for the corresponding kind.
2. The top-level `status` field of a cyclic verdict uses only `VALID-CYCLIC-PROVENANCE-LFP`, `INVALID-CYCLIC-PROVENANCE`, `INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT`, `CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS`, `CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL`.
3. The top-level `status` field of a support verdict uses only `VALID-SUPPORT-GROUNDING-LFP`, `INVALID-SUPPORT-GROUNDING`, `INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT`, `SUPPORT-GROUNDING-INVARIANT-AUDIT-PASS`, `SUPPORT-GROUNDING-INVARIANT-AUDIT-FAIL`.
4. `reason_codes` is a separate array of reasons. The string `NULLARY-INFERENCE-REQUIRES-EXPLICIT-BASE` is one of its elements, not a status.
5. `ABSENT`, `CUT-CERTIFIED`, `UNBLOCKABLE-PROTECTED-ENDPOINT` are literal values of the nested field `query_results[].minimum_internal_vertex_cut.cut_status`, not a top-level status.

6. CP-PATH-FALSIFIER, NO-ORIGINLESS-ATOM-IN-LFP, FIXED-POINT-POLICY-SUPPLIED, UNSUPPORTED-NONLEAST-FIXED-POINT, REJECTED-OVERREACH, and the other CAPS strings not named above as a specific schema field are protocol-level labels/observations. They are not implicit schema enums.

7. Every machine-verdict retains `semantic_status`: NOT-MACHINE-VERIFIED.

The words `status`, `label`, `falsifier`, and `verified` also remain recorded forms. The fence distinguishes their protocol roles; it does not attest to what they name.

1. Exact objects under attack

1.1. CP-LFP-3: cyclic least-provenance

Let $G=(V,E)$ be a finite directed graph. The set of admissible CP sorts T_3 consists of exactly eight elements:

- *FORMAL, BINDING, EVENT, GOAL*;
- *REFERENT, CAUSE, APPLICATION, MEANING*.

These are the same uppercase mathematical sort-codes as in 00 and 01. The JSON schema field `sort` stores their lowercase representatives under the fixed bijection $\text{FORMAL} \leftrightarrow \text{"formal"}$, $\text{BINDING} \leftrightarrow \text{"binding"}$, $\text{EVENT} \leftrightarrow \text{"event"}$, $\text{GOAL} \leftrightarrow \text{"goal"}$, $\text{REFERENT} \leftrightarrow \text{"referent"}$, $\text{CAUSE} \leftrightarrow \text{"cause"}$, $\text{APPLICATION} \leftrightarrow \text{"application"}$, $\text{MEANING} \leftrightarrow \text{"meaning"}$. The words `type`, `kind`, and `role_code` are not synonyms for the field `sort`. For each node s , sort $\tau \in T_3$, and atom-code a , a protected relation is defined:

$$a \in \text{Intro}_\tau(s).$$

Origin-tagged introductions are defined by the formula

$$\text{OIntro}(s) = \{(s, \tau, a) : \tau \in T_3, a \in \text{Intro}_\tau(s)\}.$$

A protected atom has exactly the triple form (s, τ, a) : s is the exact origin-id, τ is one mathematical/abstract sort-code from T_3 with a fixed lowercase JSON representative, and a is the atom-code. Permuting the coordinates, replacing the sort with node `kind` or SG `role_code`, or placing a similar string in the payload does not create this triple.

On the componentwise ordered finite lattice, define the operator

$$\Phi_G(X)_v = \text{OIntro}(v) \cup \bigcup_{u \rightarrow v} X_u.$$

It is monotone. Normative provenance is defined not by an arbitrary solution of the equations, but by the least fixed point:

$$P^\mu = \mu \Phi_G.$$

The exact CFP-3 formula is:

$$P^\mu(v) = \{(s, \tau, a) : (s, \tau, a) \in OIntro(s), s \rightarrow^* v\}$$

Equivalently, after substituting the definition of $OIntro$, the introduction condition is $a \in Intro_\tau(s)$. Here $s \rightarrow^* v$ permits a path of length zero. Therefore, a node's local introduction is already included in its own least-provenance.

1.2. SG-LFP-3: grounded support

Let N be the finite set of all protected code-ids, $S \subseteq N$ the set of abstract seed-codes, and R a finite set of positive hyperrules

$$r = (Prem(r), Head(r)).$$

In the executable embedding, the namespaces are separated: $N = B \sqcup C$, the full seed-set is $S = B$, $Prem(r) \subseteq N$, and $Head(r) \in C$. Tagged BASE and CLAIM are not merged by an identical raw id or payload.

For an ordinary hyperrule, the normative requirement is

$$Prem(r) \neq \emptyset.$$

In the abstract theorem, every leaf is a seed-code from S ; no origin field is declared for an SG base. In the executable embedding, a seed is represented by a tagged BASE-record with exact `base_id`, `role_code`, and payload. CLAIM and BASE are distinct namespaces.

Before closure, a nullary constructor must be normalized into a protected base with an exact `base_id`, for example under the convention

```
RULE-SEED:<rule-id>
```

The prefix RULE-SEED: is only an id convention, not a checker requirement. The normalizing inference must explicitly list the same exact `base_id` in `required_base_ids`. An inference for which both `premise_claim_ids` and `required_base_ids` are empty receives `status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT` and `reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE]`.

The support operator is:

$$T_S(X) = S \cup \{Head(r) : r \in R, Prem(r) \subseteq X\}.$$

A support-tree for q is a finite tree whose root is labeled q ; each internal node is expanded by one exact hyperrule and has all of its premises, not one selected subset of them; every leaf is a protected seed. GST-3 then states:

$$q \in \mu T_s \Leftrightarrow q \text{ имеет конечное support-tree с seed-листьями.}$$

The Russian text embedded in the formula means: “has a finite support-tree with seed leaves.”

For each q , the set of minimal bases is:

$$B_{\min}(q) = \min_{\subseteq} \{ \text{Leaves}(T) : T \text{ — допустимое support-tree для } q \}.$$

The Russian text embedded in the formula means: “is an admissible support-tree for.”

Minimality here means inclusion of sets of abstract seed-codes; in the executable embedding, these are exact sets of `base_id`. It does not mean truth, sufficiency outside the contract, physical source, or minimum cost.

1.3. FP-SEL-3: uniqueness of the grounded fixed point

A fixed point $X = T_s(X)$ is called support-grounded if and only if every $q \in X$ has a finite support-tree with leaves in S . FPS-3 states:

$$X = T_s(X) \wedge X \text{ support-grounded} \Leftrightarrow X = \mu T_s.$$

Consequently, if $X \neq \mu T_s$ is another fixed point, then there exists

$$q \in X \setminus \mu T_s,$$

for which no finite seed-grounded support-tree exists. Such a code receives the protocol-level label UNSUPPORTED-NONLEAST-FIXED-POINT. The word `unsupported` refers to the exact support contract, not to falsity of the payload. The mere existence of a nonleast fixed point does not mean it was selected; selection-policy becomes a readset-input only when a specific state is selected, accepted, or used.

2. Red team against CFP-3

2.1. Erroneous Broad Statement: “A Cycle Creates Nothing” (Exact Russian Source Wording: «цикл ничего не создаёт»)

It does not survive the positive control. Let $u \leftrightarrow v$, and let node u locally introduce the protected atom $\alpha = (u, \tau, a)$, that is, $\alpha \in OIntro(u)$. Then a path $u \rightarrow v$ exists, so

$$\alpha \in P^\mu(u) \text{ и } \alpha \in P^\mu(v).$$

The Russian conjunction embedded in the formula means “and.”

The cycle propagates what was introduced. The exact negative result is different:

An originless SCC does not generate a protected atom in least-provenance.

If no node reaching the SCC under consideration introduces atom-code a of sort τ , then no node in the SCC contains the corresponding origin-tagged atom in P^μ . The protocol observation is NO-ORIGINLESS-ATOM-IN-LFP; it is not a schema-status.

2.2. Locally originless SCC with an incoming path

The absence of a local introduction within the SCC is insufficient. Let $s \notin C$, $(s, \tau, a) \in OIntro(s)$, and suppose a path $s \rightarrow^* v$ exists for $v \in C$. Then $(s, \tau, a) \in P^\mu(v)$, even if every local-intro within C is empty.

Therefore, *originless* must mean the absence of an origin-tagged introduction throughout the entire incoming ancestry-cone relative to the particular atom, not merely inside the SCC. Otherwise the statement CP-NOSOURCE is false.

2.3. Path of length zero

If $(v, \tau, a) \in OIntro(v)$, then $(v, \tau, a) \in P^\mu(v)$ without a separate edge. An implementation requiring a nonzero-length path loses local introductions and falsifies CFP-3.

2.4. Edge direction

The formula uses $u \rightarrow v$ to propagate provenance from predecessor u to v . Reversing the direction computes descendants instead of ancestors. A symmetric cycle can conceal this error, so acyclic controls with one directed path are mandatory.

2.5. Origin-id, sort, and atom-code are not merged

The atoms

$$(s_0, \tau, a), (s_1, \tau, a), (s_0, \tau', a)$$

are distinct. Deduplication by payload a alone erases the origin or sort and makes the ancestry inference false. All three coordinates of the triple are mandatory. Renaming is admissible only as a bijective renaming of all protected references.

2.6. Declared provenance cannot be taken on trust

If the input JSON itself declares provenance, the checker must independently recompute $\mu\Phi_G$ and compare the exact sets. Accepting the user-supplied list as a basis turns the string *origin* into self-certification.

2.7. Changing the graph during closure

CFP-3 concerns one frozen G , one set of introductions, and one operator. Adding an edge, deleting a node, rebinding an origin-id, or changing introductions during iteration creates a different input.

In-place computation is admissible only when equivalence to the least fixed point and schedule-independence have been proved. Otherwise traversal order becomes a hidden policy-input.

2.8. Nonmonotone recursion

Deletion, a negative premise, negation-as-failure, priority, an exception, or a rule of the form

$$T(X) = A \setminus X$$

does not satisfy the positive monotone contract. Iteration from $\emptyset$ may oscillate, a fixed point may not exist, and the update order may change the result.

Such an object receives `OUTSIDE-CP-LFP-3-NONMONOTONE` or `OUTSIDE-SG-LFP-3-NONMONOTONE`, depending on the contract being attacked. It is not a counterexample to CFP-3/GST-3; the error occurs if Phase 3 nevertheless assigns theorem-status to it.

2.9. Infinite graph

The package's machine domain is finite. The finite-path formula can be extended separately, but this requires explicit conditions on the lattice, continuity, and the method of reaching the fixed point. Checker success on finite fixtures does not close a quantifier over arbitrary infinite systems.

2.10. Strict CP-PATH-FALSIFIER

CFP-3 is falsified within the unchanged contract by a theorem-falsifier with protocol-label `CP-PATH-FALSIFIER` if an object is accepted for which at least one of the following holds:

1. the computed $P^\mu(v)$ contains a triple (s, τ, a) , although $(s, \tau, a) \notin OIntro(s)$ or no path $s \rightarrow^* v$ exists;
2. $(s, \tau, a) \in OIntro(s)$ and a path $s \rightarrow^* v$ exists, but the triple (s, τ, a) is absent from $P^\mu(v)$.

A hidden oracle, a different graph, a changed CP sort, a nonmonotone operator, or an unread channel removes the premises but is not a `CP-PATH-FALSIFIER`.

3. Red team against GST-3

3.1. A nullary rule—the critical counterexample without normalization

If one permits

$$S = \emptyset, \text{Prem}(r) = \emptyset, \text{Head}(r) = q,$$

then $q \in \mu T_S$ after the operator's first application, although there is no tree with seed leaves from S . Therefore, one of two alternatives is mandatory:

- ordinary hyperrules have nonempty premises;
- every nullary constructor is prenormalized into a protected base with an exact `base_id`—for example `RULE-SEED:<rule-id>` under the optional naming convention—and the inference to q explicitly contains that same `base_id` in `required_base_ids`.

Phase 3 adopts precisely this norm. The checker does not require the prefix `RULE-SEED;`; it requires an existing exact `base_id` and an explicit reference. An empty-premise inference with neither a claim-premise nor a base-reference is an incompatible contract, not an unnoticed exception.

3.2. A hyperedge is not a set of independent edges

For the rule

$$\{p_1, p_2\} \rightarrow q$$

both premises are required. An implementation deriving q from one p_i checks a different operator. Ordinary path reachability does not replace hyperrule closure.

3.3. A cycle with a seed is grounded

In the exact executable control, BASE and CLAIM are separated:

```
BASE:  B:seed
CLAIM: C:p
CLAIM: C:q

B:seed => C:p
C:p    => C:q
C:q    => C:p
```

The first inference line has `required_base_ids: ["B:seed"]` and empty `premise_claim_ids`; the two cyclic lines use exact claim ids. Then

$$C:p, C:q \in \mu T_S, B_{min}(C:p) = B_{min}(C:q) = \{\{B:seed\}\}.$$

The cycle does not cancel the finite basis: the support tree for $C:q$ terminates at exact base `B:seed`. Therefore, the statement «всё циклическое неподдержано» (“everything cyclic is unsupported”) is false.

3.4. A cycle without a seed does not become a basis

Given

$$S = \emptyset, p \rightarrow q, q \rightarrow p$$

the least fixed point is empty. The set $\{p, q\}$ may be another fixed point, but there is no finite seed-tree for either p or q .

This conclusion uses the condition that there are no other rules/bases providing a finite grounded tree for p or q . In general: if no finite support-tree with leaves in S exists for the head-code—in particular, if only a seedless positive cycle is available—then the head-code is not in μT_S . The presence of a seedless cycle does not cancel an alternative grounded derivation if one is recorded.

3.5. A support-tree expands a rule in full

The same code may occur in multiple branches of the tree. Prohibiting a label from occurring more than once turns the tree into a DAG and may erroneously reject a finite unfolding. The restriction concerns finiteness of the tree, not global uniqueness of every label.

At the same time, a branch that repeats a cycle indefinitely and does not terminate in a seed is not a finite support-tree.

3.6. A seed is a protected constructor, not a word

The payload

```
seed
RULE-SEED: r0
q has support
```

does not add an element to S . In the executable checker, support arises only through an exact protected base record and an explicit `required_base_ids` reference from an inference. The prefix `RULE-SEED:` inside a payload or `base_id` remains a naming convention and does not itself create a base. In the abstract theorem, a leaf is required only to be an exact seed-code $s \in S$; `BASE/required_base_ids` are details of the executable embedding. Otherwise, the payload assigns itself status.

3.7. Minimal bases

$B_{\min}(q)$ must enumerate all subset-minimal leaf-sets of admissible trees. Four errors are dangerous:

1. selecting a basis with the smallest number of leaves instead of a subset-minimal basis;
2. deleting one of several incomparable minimal bases;
3. retaining a strict superset of an already found basis;
4. treating an intermediate derived code as a leaf even though it is not in S .

If nullary constructors are normalized, the protected bases created for them participate in the basis through their exact `base_id`, explicitly required by the corresponding inference. `RULE-SEED:<rule-id>` is an admissible example of a convention-id, not a mandatory prefix.

3.8. Strict SG-SUPPORT-FALSIFIER

GST-3 is falsified within the unchanged finite positive support contract by a theorem-falsifier with protocol-label SG-SUPPORT-FALSIFIER if at least one of the following is presented:

1. $q \in \mu T_s$, but no finite support-tree with seed leaves exists for q ;
2. an admissible finite support-tree for q exists, but $q \notin \mu T_s$.

An empty-premise rule without the normative normalization does not refute the corrected GST-3: such an object receives status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT and reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE].

4. Red team against fixed-point selection

4.1. Solving equations is not grounded provenance

For an originless two-node cycle

$$X_u = X_v, X_v = X_u$$

every pair $X_u = X_v = Y$ is a fixed point. The fact of fixedness alone cannot select Y .

The least fixed point with empty introductions is

$$X_u = X_v = \emptyset.$$

A nonempty Y is circular residue relative to finite-path provenance. Repetition around the cycle does not make it an origin-tagged introduction.

4.2. The greatest fixed point is not a hidden least fixed point

Choosing νT_s , an arbitrary fixed point, a stable model, or a coinductive closure may be useful under another contract. Phase 3 assigns none of those alternatives the literal value FALSE; it records the status FIXED-POINT-POLICY-SUPPLIED.

If

$$\nu T_s \neq \mu T_s,$$

then every code in the difference $\nu T_s \setminus \mu T_s$ lacks a finite seed-grounded support-tree. If $\nu T_s = \mu T_s$, there is no unsupported residue; one cannot claim that a gfp always adds extra codes.

Every exact fixed point must lie in the interval

$$\mu T_s \subseteq X \subseteq \nu T_s.$$

A violation of this inclusion for the same monotone operator is an `FP-INTERVAL-FALSIFIER`.

4.3. The least policy is recorded too

Uniqueness of the grounded fixed point explains why the finite seed-support contract selects μT_S . But the name `least`, the order formula, and the groundedness criterion are themselves part of the specification. They are not properties of the world and receive no privilege before being recorded.

Silently switching `least` to `greatest`, `any`, `stable`, `well-founded`, `first-found`, or a solver default changes the problem. If policy affects output, it belongs in the readset.

If one policy-free core requires two different policy-selected outputs, no single total function of that core alone produces both. A contrary object would be an `FP-POLICY-FALSIFIER`. The difference between outputs under different recorded policies is itself the positive control `POLICY-SENSITIVE-CODE`.

4.4. Iteration order

The normative result is defined by the mathematical μT_S , not by an accidental node order. An implementation must either use synchronous iteration from bottom or prove that its fair worklist computes the same μT_S . An unfair schedule that forever skips an enabled rule may return a strict subset and is not the least fixed point.

4.5. Strict FP-GROUNDING-FALSIFIER

FPS-3 is falsified with status `FP-GROUNDING-FALSIFIER` if, within the unchanged finite positive contract, a fixed point $X \neq \mu T_S$ exists for which every element has a finite support-tree with leaves in S .

An implementation-falsifier occurs if the checker accepts a nonleast X , localizes no $q \in X \setminus \mu T_S$, and simultaneously assigns all of X the status `support-grounded`. It becomes a theorem-falsifier `FP-GROUNDING-FALSIFIER` only if, with $X \neq \mu T_S$, a finite seed-grounded support tree has in fact been presented and accepted for every $q \in X$.

A separate `FP-RESIDUE-FALSIFIER` occurs if the nonempty residue of an accepted fixed point does not allow backward choice to continue along actually firing positive rules to a residue-premise and a finite dependency-cycle. The same finite universe, exact fixed-point equality, and unchanged rules are required.

5. Hidden readset, an unclosed chain, and regress

5.1. Readset completeness does not follow from a declaration

A procedure may read the network, clock, random seed, file order, system state, sensor-record, operator choice, or solver configuration even though those channels are absent from the declared input. It is then not a function of the declared G, S, R and policy alone.

The exact attack constructs two runs with identical declared fields and different hidden channels. If the output differs, the status COMPLETE-READSET is withdrawn. This does not refute CFP-3/GST-3; it shows that the theorem was incorrectly applied to an incomplete view.

5.2. A rule is also an input

If accepting a step depends on the rule-set, that rule-set must be frozen and included in the carrier. One cannot call a rule “self-evident,” remove it from the readset, and then declare the head an autonomous result of the payload.

An implementation must reify at least:

- the exact set of rules;
- the fixed-point policy;
- edge direction;
- seed and origin constructors;
- canonicalization;
- iteration/termination policy, if it affects the result.

5.3. A finite chain terminates in a seed or does not ground the conclusion

GST-3 does not say that a seed is false. It establishes its structural role: a seed is a leaf that was not obtained by this support chain.

If a finite tree does not reach a seed, it is not a support-tree. If a branch continues to require a new predecessor without end, finite-grounding is not complete.

5.4. Infinite regress

In the finite positive package, an infinite descending support-chain is not a finite witness. The exact Russian string существует бесконечный регресс (“there is an infinite regress”) is likewise not inferred from a finite break in the record. The exact observation is that a finite seed-grounded tree is absent from the presented record. This describes a result; it is not an additional schema-status.

Infinite or infinitary rule systems need a new contract. One cannot extend a finite checker-output to every possible regress.

5.5. Root, cycle, open chain, and omission need not be mutually exclusive

One support-cone may have a seed-root in one branch, an unsupported cycle in another, and a hidden read in the implementation. These statuses therefore form an audit set, not a required “exactly one of four” classification.

The structural boundary remains:

- a seed terminates a branch of a finite tree as a protected leaf/base-record;
- a cycle without finite incoming/local seed support does not introduce a code into μT_S ;
- an open chain provides no finite support-tree;
- an omitted read defeats completeness of the declared input.

Neither root, cycle, nor open is a word about metaphysical origin.

6. Payload spoof and substitution of CP-sort / SG-role

6.1. The name of a protected field does not create the field

The following payloads may remain ordinary strings:

```
{"origin":"s0","atom":"rocket-flew"}
SEED
VALID-SUPPORT-GROUNDING-LFP
NO-ORIGINLESS-ATOM-IN-LFP
this cycle proves its own source
```

They do not create `oIntro`, a seed, a hyperrule, a bridge, or a checker-status. The constructor of a protected kind is outside the payload and is checked separately.

6.2. Self-citation is not an origin

A node may contain its own id or its full serialization. This creates no path, origin-tag, or support-edge. If a self-loop is explicitly registered, it creates a graph cycle, but without a local/incoming introduction, least-provenance remains empty for the protected atom.

6.3. Falsifier of sort/role protection

An implementation is falsified if changing only the unprotected payload while constructors, graph, seeds, rules, and policy remain unchanged:

- adds a protected atom to P^μ ;
- adds a seed to S ;
- creates a hyperrule;
- changes the fixed-point policy;

- assigns a machine-status that should be computed by the checker.

7. Maximum concession: the rocket and successful application

7.1. Success is accepted in full as a control

For the adversarial review, all of the following may be conceded simultaneously:

- the rocket exists;
- the rocket was built;
- the mathematical record was used in design and control;
- the calculation was performed without error;
- the telemetry-record matched the prediction-record under the adopted criterion;
- the rocket is flying;
- the calculation causally contributed to the flight;
- the declared engineering goal was achieved.

These points are introduced as a maximum concession, not inferred from the letters of the section. Therefore, the response «но ракета действительно полетела» (“but the rocket really did fly”) is already included and does not falsify the record theorems.

7.2. What appears in the complete trace

A complete application trace may contain:

$$(m, b, r_{app}, o, p, c, g, j),$$

where m is a formal result, b a binding/bridge record, r_{app} an application record, o an observation/telemetry record, p an occurrence or mechanism protocol, c a fit criterion, g a goal record, and j a justification record.

After a protected introduction, $\mu\Phi_G$ propagates origin-tagged atoms along graph-edges; after bases and rules are registered, μT_S derives supported claim-codes. These are different relations. The positive CP-control receives CYCLE-PROPAGATES-INTRODUCED, not a violation of Phase 3.

7.2.1. Goal-sort, SG-base, and purpose-claim are not merged

The three executable forms are distinct:

Layer	Exact protected form	What the form does not create by itself
CP	triple $(s, GOAL, a) \in OIntro(s)$, serialized with JSON sort: "goal"	SG-base or purpose-claim
SG BASE	{ "base_id": "B:goal", "role_code": "goal", ,... }	CP origin triple
SG CLAIM	{ "claim_id": "C:purpose", ... }	goal-base; CLAIM has no sort or role_code

In the rocket control, inference `I:purpose` derives claim `C:purpose` and explicitly lists `B:goal` in `required_base_ids`. Prefixes `B:` and `C:` are id conventions, but the distinction between schema namespaces is mandatory. The CP mathematical sort *GOAL*, its JSON representative sort: "goal", SG `role_code`: "goal", and the payload word `goal` do not inherit from one another without a separate embedding.

The labels `endorsement`, `fit`, and `justification` do not automatically belong to the eight CP sorts. In SG, they may be exact claim ids or protected bases with an admissible `role_code` chosen by the contract; their English spelling does not add a new enum sort.

7.3. What success does not write backward

A later success-record does not become the earlier origin of the mathematical carrier. It creates a new node, a new path, and a new available input. A feedback cycle may propagate sensor-, control-, and goal-codes after their introduction; it does not make them originless content of the earlier formula.

It does not follow from this:

- that the rocket does not exist;
- that the flight did not occur;
- that the calculation is useless;
- that there is no causal connection;
- that the sensor is unreliable;
- that the goal is absent;
- that success "means nothing."

Each such negative statement would require a separate relation and would violate polar symmetry.

7.4. The question "why?" does not refute the computation

Goal, endorsement, fit, and justification are different target-role codes, not automatically four CP sorts. A formal procedure may propagate an already introduced CP goal-atom or derive an SG claim from exact

goal/criterion bases. It does not autonomously select these inputs from an originless CP-cycle or a seedless SG support-cycle.

The exact result is:

The rocket may fly, the calculation may match, and the control cycle may work. But binding, observation, criterion, goal, and justification enter the complete trace as distinguishable CP introductions, SG seeds, or supported claims; repetition creates neither CP output without an introduction nor SG output without seed-support.

7.5. A physical cycle is not a support-SCC

A feedback loop, an orbit, an electrical circuit, repeated program execution, and an SCC in a record graph are different forms until a mapping contract is supplied. CFP-3 does not state that a physical cycle cannot cause anything. It computes the provenance of protected codes in the presented G .

8. Performative contradiction and self-application

8.1. This document is not an exception

The present file is a carrier. Its formulas receive theorem-status only relative to the declared metatheory, encoding, and reading rules. Its strings are not the things named by *источником* (source, in the inflected Russian form), *циклом* (cycle, in the inflected Russian form), *ракетой* (rocket, in the inflected Russian form), *успехом* (success, in the inflected Russian form), or *миром* (world, in the inflected Russian form).

Phase 3 uses mathematics to establish a mathematical invariant of finite positive closure. This is not a contradiction. A contradiction would arise from simultaneously recording:

1. no formal inference under any rules is possible;
2. the following formal inference proves item 1.

There is no such target here. Internal derivability is accepted. What is denied is the originless appearance of a protected atom in least-provenance; for $X \in \text{Fix}(T_s) \setminus \{\mu T_s\}$, the absence of a finite seed-tree is established for at least one residue-element. A separate absence of a finite introduction-path witness holds for nonleast unary Φ_G .

8.2. Policy and seeds of the proof itself

The inquiry itself uses recorded rules of logic, definitions of a finite graph, a lattice order, and a fixed-point policy. They are a supplied contract, not a result of self-certification. FPS-3 does not turn the word *least* into a property of anything outside the contract.

8.3. Checker Output Remains a Record

The status

VALID-CYCLIC-PROVENANCE-LFP

means that the supplied bytes passed the exact checker under the declared contract. It assigns no truth status to the payload and does not establish an external or historical origin, physical history, or completeness of the world.

Likewise:

- a hash fixes a relation between computed bytes and a digest under the algorithm;
- a test count records the results of the given runs;
- a manifest records the listed paths and hashes;
- a freeze label does not establish an observation's date of origin;
- a self-audit does not attest to the text's external truth.

8.4. Public statement

The following exact phrase appears in the record:

«Цикл не является источником» (“A cycle is not a source”)

When this phrase appears without a subtitle, it receives UNBOUNDED-STANDALONE. The exact admissible statement is:

An originless SCC does not introduce a protected atom into least-provenance; a cycle with a local or incoming introduction propagates what was introduced, and policy is an added input when it selects, accepts, or uses a nonleast fixed point.

8.5. Historical and ontological statements

The phrases «математика возникла из языка» (“mathematics arose from language”), «чисел не существует» (“numbers do not exist”), «всё является линиями» (“everything is lines”), and «никто раньше этого не делал» (“no one has done this before”) are not premises of CFP-3, GST-3, or FPS-3. Excluding them strengthens the theorems: the result does not depend on a story about the past or on the ontology of the word *число* (number).

An author may separately declare that an observation originated independently; such a declaration may be recorded as AUTHOR-DECLARATION. It receives no theorem-status and is not required for the Phase 3 results.

A registered novelty-claim has exactly the following fields:

```

claim_id: C-NOVELTY
kind: novelty_status
novelty_status: NOT-ASSESSED
priority_status: NO-PRIORITY-CLAIM
expected_status: NOT-ASSESSED
semantic_status: NOT-MACHINE-VERIFIED
claim_sha256: sha256:6d6a54955d87d6cd35a15cb50eb4a7e3f1c20d61a50cbdeb5f30518db9f9914d

```

NOT-ASSESSED means neither первый (first), не первый (not first), nor совпадений нет (no matches). AUTHOR-DECLARATION is a protocol annotation and does not replace a novelty audit.

9. Failure conditions

9.1. Formal failure

Phase 3 requires correction if at least one of the following is presented:

- CP-PATH-FALSIFIER while preserving the exact finite positive CFP contract;
- CP-PATH-FALSIFIER in the CP-NOSOURCE specialization, CP-SCC-FALSIFIER, CP-CUT-FALSIFIER, or CP-AST-FALSIFIER while preserving their exact additional premises;
- SG-SUPPORT-FALSIFIER, SG-BASIS-FALSIFIER, or SG-HIT-FALSIFIER while preserving the exact finite positive support contract;
- FP-GROUNDING-FALSIFIER, FP-RESIDUE-FALSIFIER, FP-POLICY-FALSIFIER, or FP-INTERVAL-FALSIFIER for a fixed-point object under the same contract;
- an originless SCC receives a protected atom in $\mu\Phi_G$;
- a reachable origin-tagged introduction is absent from $\mu\Phi_G$;
- $q \in \mu T_S$ without a finite seed-tree;
- a finite seed-tree exists, but $q \notin \mu T_S$;
- a support-grounded fixed point differs from μT_S ;
- some $X \in \text{Fix}(T_S)$, $X \neq \mu T_S$, contains no residue-element lacking a finite seed-tree;
- a declared protected-endpoints CP-CUT deletes internal nodes intersecting every introducer-to-target path, but the atom remains under the unchanged deletion contract;
- a declared CP-CUT leaves a surviving introducer-to-target path, but the atom disappears;
- a declared CP-AST permits different target-codes for equal frozen core and least-provenance, even though complete factorization of the target through those two inputs is preserved.

9.2. Implementation failure

The checker requires correction if it:

- accepts declared provenance without independent recomputation;

- starts iteration from a user-supplied nonbottom seed not registered in O_{Intro} or S ;
- computes a gfp or first-found fixed point and calls it an lfp;
- produces a result that depends on an undeclared traversal order;
- reverses the direction of even one provenance-edge;
- merges atom-codes with different origin-ids or sorts;
- treats a hyperedge as a disjunction of premises;
- accepts an unmarked empty-premise rule;
- admits a negative or deleting rule as positive monotone;
- allows a payload to create a protected constructor;
- omits a reachable introduction or adds an unreachable atom;
- declares an entire arbitrary fixed point grounded merely from equality $X = T_s(X)$;
- loses one of the incomparable minimal bases or retains a demonstrably nonminimal superset;
- crashes on structurally invalid input instead of returning a bounded status.

9.3. Application failure

The theorem may remain valid but be applied incorrectly if:

- an SCC in a record graph is presented as a physical causal loop;
 - a protected atom is presented as a referent;
 - a seed is called false merely because it is not derived by this chain;
 - circular support is called a logical contradiction without a separate consistency contract;
 - a gfp is declared inadmissible in every coinductive system;
 - a finite checker is presented as a proof about every infinite recursion;
 - a locally originless SCC is called globally originless despite an incoming introduction;
 - successful application, a sensor, or a goal is covertly present in the input but declared the result of a single mathematical projection;
 - absence of a goal-record is extended to absence of a goal;
 - absence of a finite support-tree is extended to falsity of the payload;
 - no translation preserving rules, readset, CP sorts, SG role-codes, origins, seeds, and fixed-point policy is presented for real practice;
 - the title is expanded into «вся математика ложна» (“all mathematics is false”) or «никакой расчёт не работает» (“no calculation works”).
-

10. Mandatory positive and negative controls

The column below contains mixed levels only with explicit marking: schema-status or reason-code is named as a field; the other elements are checkable observations/protocol-labels, not values of the top-level status.

Control	Expected record result
One local introduction, an acyclic path to the target	status: VALID-CYCLIC-PROVENANCE-LFP; the atom is present at all reachable descendants
Originless self-loop	NO-ORIGINLESS-ATOM-IN-LFP; least provenance is empty
Originless two-node SCC	NO-ORIGINLESS-ATOM-IN-LFP; nonempty equation-solutions are marked nonleast
SCC with a local introduction	CYCLE-PROPAGATES-INTRODUCED
SCC without a local introduction but with an incoming origin	CYCLE-PROPAGATES-INTRODUCED
Identical payload under different origin-ids	Two different protected atoms
Payload contains the full serialization of an origin/seed	Protected closure does not change
Tagged executable seeded-cycle: BASE B:seed; CLAIMs C:p, C:q; B:seed $\Rightarrow$ C:p, C:p $\Rightarrow$ C:q, C:q $\Rightarrow$ C:p	status: VALID-SUPPORT-GROUNDING-LFP; for C:p and C:q, the exact minimal base-set equals {B:seed}
Only C:p $\Rightarrow$ C:q and C:q $\Rightarrow$ C:p, with no BASE and no alternative finite seed-grounded support-tree	protocol-label NO-SEED-NO-ROOT-IN-LFP; C:p,C:q are absent from μT_S
Inference with empty premise_claim_ids and required_base_ids	status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT; reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE]
A nullary constructor normalized into a BASE with an exact base_id (for example, convention-id RULE-SEED:r0) and explicitly specified with that same id in required_base_ids	Head grounded through the protected base; the prefix RULE-SEED: is not required by the checker
Hyperrule $\{p_1, p_2\} \rightarrow q$, with only p_1 available	$q \notin \mu T_S$
Two incomparable minimal seed-bases	Both are present in $B_{\min}(q)$
Nonleast fixed point of a seedless claim-cycle	UNSUPPORTED-NONLEAST-FIXED-POINT
Explicitly selected gfp policy	FIXED-POINT-POLICY-SUPPLIED
Nonmonotone recursion	OUTSIDE-CP-LFP-3-NONMONOTONE / OUTSIDE-SG-LFP-3-NONMONOTONE / OUTSIDE-FP-SEL-3-NONMONOTONE
Maximally successful rocket trace with introduced bridge/telemetry/criterion	Success is allowed; introduced codes are grounded but not retroactive

A system that accepts only convenient originless examples and does not accept a cycle with an introduced atom hard-codes a slogan in place of testing the theorem.

11. Exact claim / status / falsifier register

The global status-layer fence from § 0 applies to the entire table: only strings following an explicit `status` are literal top-level enums; `reason_codes` remain a separate field; `ABSENT`, `CUT-CERTIFIED`, `UNBLOCKABLE-PROTECTED-ENDPOINT` are only literals of the nested `cut_status`; self-claim status is specified as `expected_status`; the remaining CAPS strings are protocol labels. There are no wildcard statuses here.

Claim	Status	Exact falsifier or limit
$P^\mu(v)$ equals the set of origin-tagged introductions on finite paths $s \rightarrow^* v$	theorem-id CP-LFP-3 (CFP-3); self-claim <code>expected_status</code> : TEXTUAL-THEOREM-IN-DECLARED-METATHEORY	CP-PATH-FALSIFIER: a triple is present without $(s, \tau, a) \in OIntro(s)$ or without a path $s \rightarrow^* v$; or such an introduced reachable triple is absent from the accepted $\mu \Phi_G$
An originless SCC does not generate a protected atom in the lfp	CP-NOSOURCE · NO-ORIGINLESS-ATOM-IN-LFP	CP-PATH-FALSIFIER in the CP-NOSOURCE specialization: an atom is present although there is no corresponding introduction anywhere in the incoming cone
SCC condensation preserves least provenance	CP-SCC · SCC-CONDENSATION-PRESERVES-PROVENANCE	CP-SCC-FALSIFIER: exact condensation with preserved introductions changes root provenance
A cycle with a local or incoming introduction propagates an atom	CYCLE-PROPAGATES-INTRODUCED	A reachable introduced atom is absent from least provenance
A protected-endpoints internal cut removes an atom at the target if and only if it intersects every introducer-to-target path	protocol theorem-label CP-CUT; the nested <code>cut_status</code> equals exactly <code>ABSENT</code> , <code>CUT-CERTIFIED</code> , or <code>UNBLOCKABLE-PROTECTED-ENDPOINT</code>	CP-CUT-FALSIFIER: every path is intersected but the atom remains; or a surviving path exists and the atom disappears—with no rewiring and no new introductions
A target that factors through frozen core and P^μ does not distinguish records with equal values of those projections	CP-AST · CYCLIC-AST-SEPARATED	CP-AST-FALSIFIER: equal core/provenance, different target-codes while total factorization is preserved
$q \in \mu T_S$ iff a finite seed-grounded support-tree exists	theorem-id SG-LFP-3 (GST-3); self-claim <code>expected_status</code> : TEXTUAL-THEOREM-IN-DECLARED-METATHEORY; finite fixture status: <code>VALID-SUPPORT-GROUNDING-LFP</code>	SG-SUPPORT-FALSIFIER: one side of the equivalence holds and the other does not under the same finite positive contract

Claim	Status	Exact falsifier or limit
If no finite support-tree with leaves in S exists for a head, then the head is absent from the lfp; a pure seedless positive cycle is a special case	protocol-label NO-SEED-NO-ROOT-IN-LFP	SG-SUPPORT-FALSIFIER: the head is in μT_S although no finite seed-grounded tree exists. The presence of a separate grounded derivation derives the head and is not a falsifier
An inference without <code>premise_claim_ids</code> and <code>required_base_ids</code> is incompatible with the executable contract; normalization creates a base and an explicit reference to it	status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT; reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE] for the unnormalized case	An empty-premise inference is accepted without an explicit base, or the normalizing base is not specified in <code>required_base_ids</code>
$B_{\min}(q)$ contains all and only subset-minimal seed leaf-sets	MINIMAL-ROOT-BASES-ENUMERATED	SG-BASIS-FALSIFIER: a minimal basis is omitted, a strict superset is added, or a leaf is not a protected seed
The root disappears after seed deletion iff the disabling set intersects every minimal basis	ROOT-SEED-CUT-VALID	SG-HIT-FALSIFIER: every basis is intersected but the root remains, or a surviving basis exists but the root disappears
The support-grounded fixed point is unique and equals μT_S	theorem-id FP-SEL-3 (FPS-3); self-claim expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY; protocol-label UNIQUE-SUPPORT-GROUNDED-FIXED-POINT	FP-GROUNDING-FALSIFIER
Every fixed point lies between the least and greatest	LEAST-FIXED-POINT-FROM-BOTTOM / GREATEST-FIXED-POINT-FROM-TOP	FP-INTERVAL-FALSIFIER: an exact fixed point does not contain the lfp or is not contained in the gfp
A nonleast fixed point contains unsupported residue leading backward to a dependency-cycle	UNSUPPORTED-NONLEAST-FIXED-POINT / CIRCULAR-RESIDUE-LOCALIZED	FP-RESIDUE-FALSIFIER; special grounding-falsifier: $X \neq \mu T_S$, but every $q \in X$ has a finite seed-tree
Selection of lfp, gfp, or another semantics is a contract field	FIXED-POINT-POLICY-SUPPLIED / POLICY-SENSITIVE-CODE	FP-POLICY-FALSIFIER; implementation limit: output changes with policy, but policy is absent from the declared readset
Gfp always differs from lfp and always adds residue	REJECTED-OVERREACH	Any contract with $\nu T_S = \mu T_S$
Nonmonotone recursion is covered by CFP-3/GST-3/FPS-3	OUTSIDE-CP-LFP-3-NONMONOTONE / OUTSIDE-SG-LFP-3-NONMONOTONE / OUTSIDE-FP-SEL-3-NONMONOTONE	Application requires a separate theorem; the current statement is not accepted
A payload can assign itself an origin, seed, rule, or policy	PAYLOAD-SPOOF-REJECTED / PAYLOAD-SUPPORT-SPOOF-REJECTED / PAYLOAD-POLICY-SPOOF-REJECTED	Protected constructors are not read from the payload; an implementation that does so is falsified

Claim	Status	Exact falsifier or limit
A declared readset is complete merely because it says so	NOT-ATTESTED	Two runs with identical declared input and different output because of a hidden channel
An open or infinite chain without a seed is a finite support-tree	REJECTED	By the definition of finite tree; an infinite domain requires a new contract
Maximum rocket-success refutes Phase 3	NOT-A-FALSIFIER • ADDED-INTRODUCTIONS	Bridge, telemetry, criterion, mechanism, and goal enter as new protected records
CFP-3 proves the absence of physical causation	REJECTED-CATEGORY-TRANSFER	The result concerns path provenance codes, not physical cause
Absence of a seed proves falsity of the payload	REJECTED-POLARITY-EXPORT	GST establishes only the absence of a finite seed-tree in the declared support system
This document is excluded from its own boundary	REJECTED-SELF-EXCEPTION	Its formulas and statuses are also carrier and contract-relative records
Checker-output establishes the external world	REJECTED-SCOPE-EXPORT	The checker processes the supplied bytes; the contract contains no separate world-relation
The title is itself a theorem	For the claim registered in 00_MANUSCRIPT_RU.md, C-PUBLIC-TITLE: kind: title; expected_status: UNBOUNDED-STANDALONE. The headings of this adversarial file are not separately registered	Only the exact bounded CFP-3/GST-3/FPS-3 formula has theorem status
Phase 3 established historical priority	novelty_status: NOT-ASSESSED; priority_status: NO-PRIORITY-CLAIM	A historical claim is not a premise and requires a separate contract

12. Final red-team boundary

After all attacks, the exact result remains:

In a finite positive contract, least-provenance contains exactly the origin-tagged introductions that reach a node along finite paths. A cycle may propagate what was introduced, but an originless SCC does not introduce a protected atom. In the support component, nullary constructors are prenormalized into protected bases and ordinary rules have nonempty premises: a code belongs to the grounded least fixed point exactly when a finite seed-support tree exists. Every $X \in \text{Fix}(T_S)$ distinct from μT_S contains residue without a finite seed-tree; a separate path-grounded

statement applies to unary Φ_G . Selection of another fixed-point policy remains an added contract field.

The short public form that does not destroy the theorem is:

A cycle may repeat what was introduced. Repetition did not thereby become a source.

Even this statement remains a carrier. It does not present a source, cycle, rocket, flight, goal, or absence of meaning.