
A PROOF DOES NOT PRESENT WHAT IT PROVES

The Contract-Dependent Proof-Status Theorem and the Boundary of Formal Derivation

Kirill Shebetov · beforeword / research

Phase 5 · English reading edition · 30 August 2026

Contents

Abstract

0. Before the Word “Proof”
1. The Exact Registered Phase-5 Boundary
2. Relation to Phases 0–5
3. Typed Model
4. Exact Read Contract
5. Decoding, Strict Prefix, and Target
6. Formal Status and Three Digests
7. ADM-5: The Finite Generated-Trace Theorem
8. Minimal Separating Pair
9. PSR-5: The Proof-Status Dependency Theorem
10. The Quantifier Limits of PSR-5
11. “The Rules Are Also Proved” and the Meta-Contract
12. Inconsistency and Explosion
13. NEX-5: The Boundary of Formal Export
14. Accepted Is Not External TRUE; Rejected Is Not External FALSE
15. The Rocket: A Complete Positive Control
16. “What For?” and 100/200/300 Years
17. Strict Scope Limits
18. Exact Falsifiers
19. Runtime Status Map
20. Formal Self-Boundary
21. Claim/Status and Dependency
22. Claim Boundary
23. The Final Boundary of Phase 5

Source Status

01. THE PROOF-STATUS READSET THEOREM

A PROOF DOES NOT PRESENT WHAT IT PROVES

0. Scope of the result
1. Exact surface (P)
2. Finite typed reading contract (C)
3. Sequential validator
4. Extensional ReadSet
5. PSR-5
6. Minimal collision pair
7. ADM-5
8. Further exact counterexamples
9. Metaproof

10. Gates
11. Certificate
12. Forbidden exports
13. Extensions
14. Limits
15. Falsifiers
16. Self-boundary
17. Final Phase 5A cut

02. THE NO-AUTONOMOUS-EXPORT THEOREM

AN ACCEPTED DERIVATION DOES NOT PRESENT AN APPLICATION

0. Domain of the result
1. Fixed accepted derivation core
2. Opaque export sorts
3. Export binding
4. Abstract export function
5. Extensional export ReadSet
6. NEX-5
7. Minimal binding pair
8. Unbound is not a negative world status
9. A recorded binding does not present a referent
10. Exact counterexamples
11. Export gates
12. Executable specialization
13. Relation to an operating device
14. Extensions
15. Limits
16. Falsifiers
17. Self-boundary
18. Final Phase 5B cut

03. AN AXIOM AND A RULE DO NOT ADMIT THEMSELVES

Parser, checker, compiler, and metaproof coordinates in proof status

Abstract

0. Records are presented first
1. Exact carriers
2. Protected statuses
3. ADR-5: ADMISSION DEPENDENCY READSET
4. ReadSet form of ADR-5
5. Strict-prefix witness with unchanged lines
6. A parser is not a coordinate-free window
7. An axiom as an explicitly admitted pair

8. A rule as an explicitly admitted transformer
9. The proof checker is a coordinate
10. Compiler and runtime do not close the boundary
11. A metaproof does not admit itself
12. Positive admission graph
13. NSA-5: the no-source-free-admission theorem
14. Cycle and regress are different records with the same boundary
15. An inconsistent calculus as a positive control
16. Labels inside the payload are inert
17. The four states of admission and label
18. Protected ReadSet matrix
19. Attempts to bypass the boundary
20. Exact falsifiers
21. Failure classes
22. Self-application
23. Limit of the result
24. Final cut

04. A FORMAL DERIVATION DOES NOT PRESENT A ROCKET

Binding, application, goal, Mars, and meaning after 100/200/300 years

Abstract

0. The initial stop at words
1. Typed carriers
2. Opaque marks an exact boundary
3. Proof core and its digest
4. Exact binding model
5. A recorded binding does not present a bound object
6. BCR-5: the binding-coordinate ReadSet theorem
7. Two exact counterbindings
8. Null binding
9. NOC-5: no export without a constructor
10. Complete chain of conditional statuses
11. The four derivation/application states
12. The four goal/endorsement states
13. Truth (Истина) as an intermediate payload label
14. A rocket as an opaque code
15. Flight as a relation code
16. Mars as a destination code
17. Maximal engineering concession
18. Positive controls
19. Counterbindings

20. A goal is not derived from a trajectory
21. The “What For?” question (для чего?)
22. No negative from silence
23. Horizon records 100/200/300
24. HRZ-5: time is not a type constructor
25. Repetition does not add a coordinate
26. 100/200/300 years and a rocket
27. Destination, goal, and meaning do not form a ladder
28. The compiler/checker layer as a record
29. Falsifiers
30. Limits
31. Self-application
32. Final cut

05. ADVERSARIAL TESTING AND SELF-APPLICATION OF PHASE 5

PROOF DOES NOT PRESENT THE PROVED

0. Purpose and zero concession to the label
1. Exact object of Phase 5
2. Admission gates
3. The three results to be attacked
4. Red-team status discipline
5. Attack A — same bytes, two parser-contracts
6. Attack B — tokenization variant
7. Attack C — whitespace and normalization
8. Attack D — same bytes, different axioms
9. Attack E — same bytes, different rules
10. Attack F — extension with a different side condition
11. Attack G — one parse, different goals
12. Attack H — accepted derivation without binding
13. Attack I — one accepted proof, two bindings
14. Attack J — two truth assignments
15. Attack K — inconsistent axioms and explosion
16. Attack L — paraconsistent variant
17. Attack M — circular derivation
18. Attack N — future-line citation
19. Attack O — self-label spoof
20. Attack P — certificate spoof
21. Attack Q — checker as a proving machine
22. Attack R — compiler objection
23. Attack S — proof-assistant objection
24. Attack T — soundness theorem

- 25. Attack U — categorical semantics
- 26. Attack V — mathematical effectiveness
- 27. Attack W — physical rocket
- 28. Attack X — same rocket, two purposes
- 29. Attack Y — historical origin of mathematics
- 30. Attack Z — “quantity exists without a record”
- 31. Attack AA — human intuition
- 32. Attack AB — universal agreement
- 33. Attack AC — successful repetition
- 34. Attack AD — recorded test-result lines
- 35. Attack AE — hash equality
- 36. Attack AF — digital signature
- 37. Attack AG — title authority
- 38. Attack AH — theorem-label self-application
- 39. Attack AI — checker-name self-application
- 40. Attack AJ — test-name self-application
- 41. Attack AK — manifest self-application
- 42. Attack AL — self-hash
- 43. Attack AM — self-audit claim registry
- 44. Attack AN — author attribution
- 45. Attack AO — predecessors and adjacent works
- 46. Exact falsification gate for PSR-5
- 47. Exact falsification gate for ADM-5
- 48. Exact falsification gate for NEX-5
- 49. Implementation falsifiers
- 50. Required positive controls
- 51. Runtime fences
- 52. Self-application matrix
- 53. Claim registry
- 54. Red-team result
- 55. Surface, digest, and registry attacks
- 56. The final boundary of Phase 5

Abstract

Fix a finite protected surface record

$$P = (\text{surface_id}, (\ell_0, \dots, \ell_{n-1}))$$

and an exact reading contract

$$C = (id_C, \delta, A, G, E, t, m_C)$$

under a common executable policy ω . Lines are already presented by the input structure; the runtime does not segment raw bytes. The contract supplies a partial exact decoding—line-index/text $\rightarrow$ opaque statement code—along with admitted axioms, a complete table of positive ground rules, `enabled_rule_ids`, and a target code. A code is admitted either as an axiom or as the conclusion of an enabled rule whose premise codes occur among previously accepted lines. The last accepted decoded line must match the target.

Within the declared **ADM-5** contract, the following status equivalence is registered: `proof_status: ACCEPTED` corresponds to a finite generated trace with `TARGET-ACCEPTED` and exact line-level witnesses. The trace is output, not additional input. **PSR-5** records that if one surface receives different statuses under contracts whose protected projections differ only in `enabled_rule_ids`, a surface-only selector cannot return both statuses. Proof status is not contained autonomously in the lines; the distinguishing field belongs to the readset. **NEX-5 rules out the second export.** If two typed completions have the same accepted (P, C) , where $ADM_C(P) = 1$ and `proof_status = ACCEPTED`, but different referent, truth-assessment, application, goal, or meaning statuses, a selector reading only (P, C) cannot correctly return both results. An accepted derivation remains a relation within an exact contract. It does not present what is named beyond that contract. The historical sentence “mathematics arose from language” is not used as a premise. Phase 5 records the following dependency within its declared contract: a proof candidate receives no status without a read contract, and a contract-relative status creates no external target. **Keywords:** record boundary; proof candidate; read contract; enabled rule; strict prefix; generated trace; proof status; meta-contract; explosion; binding; application; goal; meaning; self-application.

0. Before the Word “Proof”

First, the surface is presented:

```
a
a -> b
b
```

Before a reading contract is supplied, it has not been established that:

- the presented `line_text` values are logical formulas;
- `a` and `b` are formula codes;
- `->` is an operator;
- the second line is an implication;
- the third line follows from the first two;
- the last line is a target;
- the surface is a proof;
- any code names anything outside the record.

The words `формула`, `правило`, `следует`, `истина`, `математика`, and `доказательство` (formula, rule, follows, truth, mathematics, and proof) likewise first enter as written forms. Their appearance does not perform the operation they name.

The executable object already contains `surface_record.lines`. This is a structural input boundary, not the result of parsing raw bytes. The checker does not assert that a line is a proposition; the exact decoding contract only maps a line index, exact `line_text`, and opaque statement code.

The zero boundary is:

surface occurrence $\neq$ formal admission $\neq$ external establishment .

The line `это доказательство` (“this is a proof”) adds a surface line. It does not add decoding, a rule table, enabled ids, a target, binding, application, or a goal.

1. The Exact Registered Phase-5 Boundary

The formal Phase-5 boundary is registered for the **autonomous proof claim**:

a finite sequence of signs allegedly contains proof status in itself and, as a consequence, presents a referent, external truth assessment, named state, application, purpose, or meaning without a separate transition.

The claim contains two hidden exports:

1. surface $\rightarrow$ proof status without an exact read contract;
2. accepted derivation $\rightarrow$ external target without a typed bridge.

PSR-5 rules out the first. NEX-5 rules out the second. ADM-5 preserves the internal positive result:

$$(P, C) \mapsto (s_C, W_C, \tau_C)$$

where, when all declared checks are satisfied, the runtime fields are `proof_status`, `trace`, and `terminal_witness`.

The generated trace is not an additional premise.

Phase 5 does not assert that “computation is impossible.” Within the declared Phase-5 meta-contract, the following distinction is registered:

$$(P, C) \mapsto \textit{FormalStatus}$$

can be exact, whereas

$$P \mapsto \{\text{proof-status, referent, external truth, application, purpose, meaning}\}$$

is not defined without additional coordinates. Under NEX-5 and its declared separating class, the autonomous-transfer claim receives the registered refutation status. The separately supplied reading rule is not derived from that status. The sentence “all mathematics is false” remains a written form without a declared evaluation contract.

2. Relation to Phases 0–5

Phase 0 required a bridge for the formal, application, and goal coordinates. Phase 1 moved the distinguishing channel into the input. Phase 2 localized origin among ancestors. Phase 3 tied positive membership to a finite support tree and separated a cycle from introduction. Phase 4 recorded a family-relative absence status under its declared predicate and included the boundary in the readset.

Phase 5 applies the same control to the label `доказательство` (“proof”):

```
surface_record.lines
-> partial exact decoding
-> opaque code sequence
-> strict-prefix admission
-> target match
-> ACCEPTED under C
-> [отдельный bridge]
-> external target-status
```

The preserved bracketed source phrase reads “[a separate bridge].”

Every arrow through `ACCEPTED` belongs to *C*. The last arrow does not arise from the preceding ones.

Compatibility with Phase 4 is exact. CBE-4 may have an accepted derivation under a declared meta-contract. Phase 5 does not repeal the family-relative theorem. It prohibits only a new renaming:

```
code-absence established in declared family
therefore
named absence established outside that family
```

The internal result is preserved; the unwritten external export is cut off.

3. Typed Model

Fix distinct types:

SurfaceRecord, Line, StatementCode, RuleId, FormalStatus, Referent, TruthAssessment, Application, GoalRecord, MeaningRecord.

Matching written forms do not fuse types. The opaque code `s:B` does not become a referent. The target code `s:MARS` does not become anyone's goal.

The surface is:

$$P = (s, L), L = (\ell_0, \dots, \ell_{n-1}), 1 \leq n \leq 32.$$

Each $\ell_i = (i, \text{line_text}_i)$ is already a member of the protected input. An index denotes position, not time, causality, or authority. Newline segmentation and Unicode normalization are not performed within the runtime; the exact contract requires `unicode_normalization: none`.

Let $Q \subseteq \text{StatementCode}$ be a finite exact universe of opaque codes. Under total decoding:

$$D_C(P) = (q_0, \dots, q_{n-1}) \in Q^n.$$

Payload is inert metadata. Writing `TRUE`, `WORLD`, or `PROOF` in payload does not change protected status.

4. Exact Read Contract

The reading contract is:

$$C = (id_C, \delta, A, G, E, t, m_C),$$

where:

- id_C is the diagnostic reading-contract id;
- δ is a partial exact decoding from indexed lines into Q ;
- $A \subseteq Q$ is `axiom_codes`;
- $G: R \rightarrow (P(Q) \setminus \{\emptyset\}) \times Q$ gives the complete ground-rule definitions;
- $E \subseteq R$ is `enabled_rule_ids`;
- $t \in Q$ is `target_code`;
- m_C is inert payload.

If $G(r) = (S_r, h_r)$, the premise S_r is a **set** of opaque codes. The order and multiplicity of premises play no role in the executable specialization. A rule id operates only when $r \in E$. The presence of a definition in G is not the same as enablement in E .

The common policy ω fixes partial line-index decoding, deterministic left-to-right validation, positive ground-set premises, an axiom-first/lexicographic-rule witness, and the final-target rule.

Runtime well-formedness and relational validation over the structural Draft 2020-12 schema require $A \subseteq Q$, $E \subseteq R$, $t \in Q$, $S_r \subseteq Q$, $h_r \in Q$, sorted unique protected ids, and an exactly supported ω . The structural schema alone does not check every sorting, cross-reference, and supported-policy gate. Nullary introductions are placed in A .

The word `contract` does not satisfy these gates. Schema validation is a separate procedure over an exact serialization.

5. Decoding, Strict Prefix, and Target

Contract decoding contains records `{line_index, line_text, statement_code}`. The checker traverses lines from left to right. At each reached index, it first checks the exact mapping. If the mapping is absent, the checker stops:

```
proof_status: UNPARSED
terminal_witness.kind: UNMAPPED-SURFACE-LINE
```

If a mapping exists, the checker immediately checks admission of that line. An earlier `LINE-NOT-ADMITTED` terminates the run, so a later unmapped index is never reached and does not change the status to `UNPARSED`.

On a fully traversed, total-decoding path $D = (q_0, \dots, q_{n-1})$, the strict prefix is:

$$H_i = \{ q_j \mid 0 \leq j < i \text{ и line } j \text{ уже accepted} \}.$$

The preserved source-form condition reads: “and line j has already been accepted.”

The exact admission condition is:

$$Good_i(C, P) \Leftrightarrow q_i \in A_C \vee \exists r \in E_C : G_C(r) = (S_r, q_i) \wedge S_r \subseteq H_i.$$

If `Good` does not hold, the checker stops:

```
proof_status: REJECTED
terminal_witness.kind: LINE-NOT-ADMITTED
```

If several rules are eligible, the runtime selects the lexicographically least rule id. For each premise code, the trace points to the earliest accepted line occurrence. This is a deterministic witness policy, not a premise order.

Target acceptance is:

$$TargetOK_C(D) \Leftrightarrow q_{n-1} = t_C.$$

It is the last **decoded code**, not its display payload, that is compared with the target.

6. Formal Status and Three Digests

The main predicate is:

$$ADM_C(P) \Leftrightarrow D_C(P) \text{ total} \wedge \forall i < n: Good_i(C, P) \wedge q_{n-1} = t_C.$$

The exact runtime field is:

```
proof_status: ACCEPTED | REJECTED | UNPARSED
```

REJECTED means a line-admission failure or a final target mismatch. UNPARSED means an unmapped surface line. They are not external negative statuses.

The `proof_readset_checker` namespace distinguishes three exact digests:

1. `surface_sha256` — a digest of the protected surface record `{surface_id, lines}`;
2. `artifact_sha256` — a digest of the canonical parsed artifact value, not arbitrary file bytes;
3. `derivation_core_sha256` — a digest of the exact canonical projection fields that determine derivation status.

The separate `phase5_self_audit` namespace reuses the field spelling `artifact_sha256` solely for SHA-256 of the exact raw fixture file bytes. This is a file policy freeze, not the canonical parsed-value digest in the proof-readset verdict. The namespace is supplied by the surrounding record contract; the two fields cannot be substituted for one another.

The exact derivation core is:

```
surface_sha256
decoding
axiom_codes
rules[{rule_id,premise_codes,conclusion_code}]
enabled_rule_ids
target_code
```

Changing canonical artifact metadata may change `artifact_sha256` without changing `derivation_core_sha256` or status. Changing `enabled_rule_ids` changes the canonical core object even if `surface_sha256` remains the same; the checker recomputes the digest. Equality of SHA-256 for two different core objects would mean a hash collision and is not derived from the model. In the published minimal pair, the two recomputed digests are in fact different. A hash reports a digest code of a canonical object under the declared algorithm; it is not an injective identity proof, proof status, or external truth status.

7. ADM-5: The Finite Generated-Trace Theorem

Let $W_C(P) = (T, \tau)$ be the generated output of a single contract evaluation: a finite trace T and terminal_witness τ . The runtime does not receive W as input and does not accept a separate certificate file.

The predicate $\text{CertOK}_C(P, W)$ in the declared metatheory means that the trace matches the left-to-right accepted prefix; each step has either an **AXIOM** or an enabled-rule witness; premise-witness indices refer to earlier accepted lines; and τ is the exact first terminal kind.

ADM-5

For runtime-valid, well-formed finite (P, C) :

$$\text{ADM}_C(P) \Leftrightarrow \text{CertOK}_C(P, W_C(P)) \wedge \tau.\text{kind} = \text{TARGET} - \text{ACCEPTED}.$$

Proof Within the Declared Meta-Contract

If $\text{ADM}_C(P)$, the total decoded sequence is finite, every **Good** holds, and the last code equals the target. The deterministic axiom-first/least-rule policy constructs a finite trace, and the terminal kind is **TARGET-ACCEPTED**.

Conversely, **CertOK** and **TARGET-ACCEPTED** fix total decoding, admission of every line, and final target equality. These are the conjuncts of $\text{ADM}_C(P)$. $\square$

ADM-5 is a positive control. The generated trace can be checked as a reference certificate, but it is not hidden input.

8. Minimal Separating Pair

The protected P_0 is the same:

```
a
a → b
b
```

The `surface_record`, `statement_codes`, `decoding`, `axiom_codes`, `complete` rules, `target_code`, `payload`, and `global` policy are fixed. The diagnostic `reading_contract_id` differs as required by the schema, but it does not belong to the derivation core.

The exact decoding, where $q_A, q_{A \rightarrow B}, q_B$ denote the three opaque ids listed immediately after the formula, is:

$$\delta(0, a) = q_A, \delta(1, a \rightarrow b) = q_{A \rightarrow B}, \delta(2, b) = q_B.$$

```
q_A      = S:A
q_A→B    = S:A_IMPLIES_B
q_B      = S:B
```

The sign $\rightarrow$ is inert within `line_text`. The whole line is mapped to the opaque `S:A_IMPLIES_B`; no operator semantics is extracted from its shape.

$$Q = \{q_A, q_{A \rightarrow B}, q_B\}, A = \{q_A, q_{A \rightarrow B}\}.$$

The complete ground-rule definition is the same:

$$G(r_{MP}) = (\{q_A, q_{A \rightarrow B}\}, q_B), t = q_B.$$

The only protected delta is:

$$E_{enabled} = \{r_{MP}\}, E_{disabled} = \emptyset.$$

The enabled run is:

```
proof_status: ACCEPTED
terminal_witness.kind: TARGET-ACCEPTED
enabled_rule_ids: [R:MP]
```

The generated rule step points to the earliest occurrences of both premise codes at indices 0 and 1.

The disabled run is:

```
proof_status: REJECTED
terminal_witness.kind: LINE-NOT-ADMITTED
enabled_rule_ids: []
```

surface_sha256 is the same. The comparison returns readset_delta_fields: [enabled_rule_ids] and READING-CONTRACT-REQUIRED-FOR-PROOF-STATUS. Only the enabled projection distinguishes the protected derivation status.

9. PSR-5: The Proof-Status Dependency Theorem

Let

$$\sigma: SurfaceRecord \times Contract \rightarrow FormalStatus$$

be a deterministic status function.

PSR-5

If

$$\sigma(P, C_0) \neq \sigma(P, C_1),$$

then there is no surface-only selector f that simultaneously returns both exact statuses:

$$\nexists f: SurfaceRecord \rightarrow FormalStatus: f(P) = \sigma(P, C_0) \wedge f(P) = \sigma(P, C_1).$$

Proof Within the Declared Meta-Contract

The function f receives the same argument P , and therefore returns one value. The premise requires two different values. One value cannot equal two distinct status codes. $\square$

PSR-READSET-5

If the contracts differ only in coordinate E , recorded in the runtime field `enabled_rule_ids`, and status differs, every correct procedure must read E or exact equivalent information.

PSR-NO-INTRINSIC-BIT-5

On the declared contract class $\{C_{ON}, C_{OFF}\}$, there is no contract-free predicate that extracts the correct proof status from P_0 alone. Reading the lines again does not recover the disabled bit. The word очевидно (“obvious”) does not enable it.

10. The Quantifier Limits of PSR-5

The separating pair proves readset necessity **within the declared model and declared contract class**. It is not a quantifier over every object ever named by the word математика (“mathematics”). This preserves the force of the argument while preventing the same unsupported export that the work rules out.

One may not infer:

два exact contracts различили один surface
therefore
установлено утверждение о каждом возможном formalism

The preserved source-form lines read: “two exact contracts distinguished one surface; therefore, a statement about every possible formalism has been established.”

One may infer:

For every contract class containing a separating pair with one P and different required statuses, proof status is not a function of P alone on that class.

This is a theorem schema with an exact premise. If a new family contains a separating pair, the result applies. If no family is presented, the word вся (“all”) does not create one.

The title thereby receives the following contract assignment:

proof as an autonomous property of lines is refuted on every class in which unchanged lines receive different statuses because of a distinguishing read contract.

11. “The Rules Are Also Proved” and the Meta-Contract

Let C_0 contain the rule id r . A surface Q_1 ends in the code `JUSTIFY(r,CO)`. To admit Q_1 , a meta-contract C_1 is required:

$$ADM_{C_1}(Q_1).$$

This adds an accepted derivation under C_1 . It neither removes C_1 nor places r in the original surface.

If the rules of C_1 require another justification, Q_2, C_2 are introduced. A finite chain is:

$$(Q_m, C_m) \rightarrow \dots \rightarrow (Q_1, C_1) \rightarrow (r, C_0).$$

META-BASE-5

For a finite meta-chain:

1. the last presented C_m is admitted as the stopping admission of the current audit; or
2. the requirement for C_{m+1} remains unrepresented within the chain.

A finite nonempty sequence has a last index. If there is no next requirement, the chain has stopped at an admission. If there is a requirement but the next pair is absent, the requirement is open. There is no third finite case. ▫ Formal practice may honestly stop at axioms, a rule table, a trusted kernel, or an execution contract. Within the declared meta-contract, the claim that “a chain of proofs eliminated its own contract” receives the refutation status. An infinite regress is not required for PSR-5. One separating pair suffices.

12. Inconsistency and Explosion

Let Q contain distinct polar codes q^+, q^- . Their joint admission does not itself create a conflict policy.

Explosive ground rules may include:

$$G(r_z) = (\{q^+, q^-\}, z)$$

for every declared $z \in Q$. If $r_z \in E$, the code z receives a strict-prefix witness. A paraconsistent contract may have the same Q, A, G, t , but $r_z \notin E$. Then the same final z is rejected in the absence of another witness. This is another PSR separating pair: the enabled id changes, not the lines. Explosion establishes reachability of an arbitrary **declared code** under exact rules. It does not present an arbitrary named fact.

Thus:

accepted after explosion $\neq$ external establishment.

The more broadly a formal rule admits outputs, the more evident the typed boundary; the payload of the result does not change its origin.

13. NEX-5: The Boundary of Formal Export

Fix a target sort

$X \in \{Referent, TruthAssessment, Application, GoalRecord, MeaningRecord\}.$

A typed completion is:

$$Z = (P, C, \beta_x, x),$$

where β_x is a binding, evaluation, or application bridge, and x is the target coordinate.

The formal projection is:

$$pr_{PC}(Z) = (P, C).$$

The target procedure is:

$$\chi_x(Z) \in TargetStatus_x.$$

NEX-5

Fix an accepted core: $ADM_C(P) = 1$ and runtime `proof_status = ACCEPTED`. If admissible completions Z_0, Z_1 of the same accepted (P, C) satisfy

$$pr_{PC}(Z_0) = pr_{PC}(Z_1)$$

and

$$\chi_x(Z_0) \neq \chi_x(Z_1),$$

then there is no selector e_x reading only (P, C) and correctly returning both target statuses:

$$\nexists e_x : e_x(P, C) = \chi_x(Z_0) \wedge e_x(P, C) = \chi_x(Z_1).$$

Proof: the projection is the same, so the single-valued e_x returns one value; the required target statuses are different. □ NEX-5 does not prohibit a bridge. It proves that the distinguishing bridge belongs to the readset.

The executable specialization uses only the record

$$B = (id_B, id_C, t, c_B, a_B, g_B).$$

The final three coordinates are recorded in the runtime fields `claim_code`, `application_code`, and `goal_code`.

The checker reports the co-occurrence of B with an accepted trace and compares binding coordinates. It does not establish a referent or the validity of the relation. The executable BCR-5 is therefore a finite witness of readset necessity, while the more general NEX-5 remains a textual theorem under `semantic_status: NOT-MACHINE-VERIFIED`.

14. Accepted Is Not External TRUE; Rejected Is Not External FALSE

The boundary must be stated without euphemism:

$$VALID - DERIVATION \neq TRUE,$$

$$INVALID - DERIVATION \neq FALSE.$$

The tokens on the right are presented here only as labels commonly used for an illicit status export. The Phase 5 runtime does not use them as unqualified external verdicts.

The exact first record is:

```
proof_status: ACCEPTED
trace_scope_status: DECLARED-FINITE-TRACE-AND-CONTRACT-ONLY
```

The exact second record is:

```
proof_status: REJECTED
terminal_witness.kind: LINE-NOT-ADMITTED | TARGET-LINE-MISMATCH
trace_scope_status: DECLARED-FINITE-TRACE-AND-CONTRACT-ONLY
```

A rejected derivation does not make the payload externally false. An accepted derivation does not make the payload externally true. External truth assessment requires β_T .

Even a formula valid in every member of an exact model class M receives a bounded status:

$$\forall M \in M : Sat(M, u).$$

The word $\forall$ (“all”) does not extend M , and a formula code does not present a structure.

15. The Rocket: A Complete Positive Control

The following chain is allowed:

```
formal calculation
-> translation specification
-> controller build
-> implementation
-> sensor protocol
-> telemetry
-> flight criterion
-> application status
```

Each transition may be executable and reproducible. Phase 5 does not deny this. But calculation status has type *FormalStatus*. The controller artifact, telemetry, and criterion have other types. Their connections belong to the application bridge.

If telemetry satisfies the exact criterion, the following status is admissible:

```
FLIGHT-CRITERION-SATISFIED-UNDER-DECLARED-TELEMETRY-CONTRACT
```

This is a bounded, strong result. It is not retroactively placed inside the mathematical lines. Consider the surface:

```
ракета полетела
```

The preserved surface reads “the rocket flew.” It remains a surface until a declared decoding and application contract are supplied. The word `ракета` (“rocket”) does not present a rocket referent. A working implementation adds records and relations. It does not prove that proof status was an autonomous property of the written form.

16. “What For?” and 100/200/300 Years

The formal target t_C and a purpose record $g \in GoalRecord$ are different.

The code `REACH-MARS` may be a target and may be accepted. This does not establish:

- whose goal it is;
- why it was chosen;
- whether it should continue;
- what meaning is assigned to it;
- whether the flight criterion answers `для чего` (“what for”).

A purpose completion requires at least:

```
goal bearer/id
goal record
adoption relation
scope/time
continuation/completion criterion
```

Consider the lines:

```
100 лет
200 лет
300 лет
```

The preserved lines read “100 years,” “200 years,” and “300 years.” They are finite surfaces. Numerals do not create a duration, future state, purpose, or meaning. A temporal constructor enters as separate input. A repeated calculation may repeat the formal output. It does not add a goal coordinate. The code `БЕССМЫСЛЕННО` (“MEANINGLESS”) may be admitted by a rule. That is membership of an opaque code, not absence of meaning. Nonmembership of a meaning code is likewise not a negative meaning status: the polarity boundary from Phase 4 remains in force.

The exact boundary is:

A rocket may satisfy a flight criterion. A derivation may satisfy a formal criterion. Neither status contains an answer to “what for” unless a purpose or meaning coordinate is introduced separately.

17. Strict Scope Limits

The admission gates are:

- protected $P = (\text{surface}_i, d, \text{lines})$ with 1–32 exact lines;
- finite Q, A, G, E , with no more than 32 statement codes and 64 rules;
- partial exact decoding by line index/text;
- every enabled id is defined;
- positive ground **set** premises, without order or multiplicity;
- prior accepted lines only;
- the final decoded line equals the target;
- deterministic axiom-first/lexicographic-rule witness;
- supported beforeword.proof-readset.v1 policy;
- for NEX-5, a typed target and separating completions.

Outside the gates are:

- actually infinite derivations;
- raw-byte segmentation and silent Unicode normalization;
- nondeterministic decoding without a declared selection policy;
- cyclic proof graphs;
- oracle admission;
- probabilistic status;
- defeasible or revision rules;
- implicit rule schemas.

Such an object requires an extended model. It is not a counterexample to unchanged Phase 5.

The runtime fences are:

```
semantic_status: NOT-MACHINE-VERIFIED
trace_scope_status: DECLARED-FINITE-TRACE-AND-CONTRACT-ONLY
world_export_status: NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER
historical_origin_status: NO-HISTORICAL-ORIGIN-ATTESTATION
```

Negative words in the fences are protocol codes. They do not present absence outside the checked field.

18. Exact Falsifiers

- **ADM-SOUNDNESS-FALSIFIER:** the generated trace has `TARGET-ACCEPTED` despite an unmapped line, a bad prefix, a missing enabled rule, or a target mismatch.
- **ADM-COMPLETENESS-FALSIFIER:** $ADM_C(P)$ holds, but the deterministic generated trace does not end in `TARGET-ACCEPTED`.
- **PSR-FUNCTION-FALSIFIER:** a single-valued surface-only f equals two distinct required statuses.
- **PSR-READSET-SPOOF:** the ON and OFF runs are distinguished without reading the enabled id or equivalent information.
- **MINIMAL-PAIR-DRIFT:** in addition to `enabled_rule_ids`, a `protected` decoding, `axiom_codes`, `rules`, or `target_code` changes.
- **ARROW-SEMANTIC-SPOOF:** the glyph $\rightarrow$ receives an operator role outside the declared opaque decoding.
- **CORE-HASH-CONFLATION:** `artifact_sha256` is passed off as `derivation_core_sha256`.
- **STRICT-PREFIX-SPOOF:** the current or a future line is used as an earlier witness.
- **DET-FALSIFIER:** identical core/checker/environment values return different deterministic statuses.
- **META-BASE-FALSIFIER:** a finite chain has no last presented index.
- **EXPLOSION-SPOOF:** an arbitrary code is admitted without enabled explosion or another witness.
- **NEX-PROJECTION-FALSIFIER:** a formal-only selector returns two distinct completion statuses for one (P, C) .
- **REFERENT-EXPORT-SPOOF:** an opaque code is promoted to a referent without a binding.
- **APPLICATION-EXPORT-SPOOF:** formal acceptance is promoted to an application status without a bridge.
- **PURPOSE-EXPORT-SPOOF:** a target code is promoted to purpose or meaning without a coordinate.
- **PAYLOAD-SPOOF:** protected status changes solely because of the writing in payload.

A theorem falsifier operates only under unchanged gates. Gate drift creates a different object.

19. Runtime Status Map

The exact overall statuses are:

```
VALID-PROOF-READSET-ANALYSIS
INVALID-PROOF-READSET-RECORD
INCOMPATIBLE-PROOF-READSET-CONTRACT
PROOF-READSET-INVARIANT-AUDIT-PASS
PROOF-READSET-INVARIANT-AUDIT-FAIL
```

The exact `proof_status` values are:

ACCEPTED
REJECTED
UNPARSED

The sole reader mapping is:

Exact runtime token	Expanded reading
ACCEPTED	all lines decoded/admitted; the final accepted code equals the target
REJECTED	the first reached line is mapped but not admitted; or every line is reached/admitted but the final target gate is not satisfied
UNPARSED	the first reached line is absent from the declared decoding; an earlier rejection would already have stopped traversal

The expanded phrases are not machine fields.

The exact `terminal_witness.kind` values are:

UNMAPPED-SURFACE-LINE
LINE-NOT-ADMITTED
TARGET-LINE-MISMATCH
TARGET-ACCEPTED

The exact pairwise comparison values are:

READING-CONTRACT-REQUIRED-FOR-PROOF-STATUS
NO-PROOF-STATUS-DIVERGENCE-IN-THIS-PAIR
BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS
NO-EXPORT-STATUS-DIVERGENCE-IN-THIS-PAIR

The exact binding-record co-occurrence values are:

EXPLICIT-BINDING-RECORD-COCCURS-WITH-ACCEPTED-TRACE
EXPLICIT-BINDING-RECORD-WITHOUT-ACCEPTED-TRACE

Overall status is not `proof_status`. `proof_status` is not a terminal kind. A co-occurrence status does not assert validity of a binding.

The public title without subtitle and definitions receives:

UNBOUNDED-STANDALONE

20. Formal Self-Boundary

This manuscript is a finite surface P_M . The words `теорема`, `доказательство`, and `опровергнуто` (theorem, proof, and refuted) grant it no exception.

Under the declared manuscript meta-contract, the following can be checked:

- typed definitions;
- the ADM-5 derivation;
- the PSR-5 separating pair;
- the NEX-5 projection argument;
- runtime/schema consistency;
- executable fixtures.

An admissible reader description is that the derivation of this manuscript can be checked under the declared meta-contract. This sentence is not a runtime machine field and does not present an external referent, historical origin, priority, or authority.

If the manuscript required autonomous external export, it would violate NEX-5. If it denied internal admission, it would destroy ADM-5. The self-boundary preserves both:

exact internal admission retained ,
unwritten external export rejected .

The sentence “Phase 5 refutes all mathematics” without a definition of математика (“mathematics”) receives UNBOUNDED-STANDALONE. The exact result is:

*No surface contains contract-free proof status on a class of contracts that distinguishes its status;
no accepted formal status itself presents a referent, external truth assessment, application,
purpose, or meaning under separating completions.*

21. Claim/Status and Dependency

```
C-PUBLIC-TITLE
expected_status: UNBOUNDED-STANDALONE
C-THEOREM-ADM-5
C-THEOREM-PSR-5
C-THEOREM-NEX-5
expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY
```

This is an exact excerpt from the frozen self-audit registry. The expressions autonomous proof-status and autonomous external export are conceptual expositions of the PSR-5/NEX-5 results, not claim ids, status values, or objects within self_claims_phase5.json; the self-audit assigns them no separate machine status.

No external author, school, journal, or term is a premise.

```
PREDECESSORS-AS-PREMISES: NONE
INTERNAL-PHASE-CONTRIBUTION: ADM-5 + PSR-5 + NEX-5 + META-BASE-5
EXTERNAL-COMPARISON: NOT-USED-AS-FOUNDATION
NOVELTY-STATUS: NOT-ASSESSED
PRIORITY-STATUS: NO-PRIORITY-CLAIM
HISTORICAL-ORIGIN-CLAIM: NOT-USED-AS-PREMISE
```

Dependency status establishes neither the presence nor the absence of predecessors. It reports only that external texts do not belong to the derivation premises.

22. Claim Boundary

Established Within the Declared Model

1. Formal admission is defined for (P, C) , not for P alone.
2. ADM-5 connects acceptance to a finite generated trace and terminal witness.
3. The minimal pair changes only `enabled_rule_ids` and changes status.
4. PSR-5 excludes a correct surface-only selector on a separating class.
5. The enabled-rule field belongs to the readset.
6. The arrow in the minimal pair is inert and mapped to an opaque code.
7. The artifact hash and derivation-core digest are distinguished.
8. A finite meta-chain stops at an admission or an open requirement.
9. Explosion derives a declared code only through an enabled ground rule.
10. NEX-5 excludes formal-only export under separating completions.
11. A referent, truth assessment, application, purpose, and meaning require bridges.
12. A flight criterion is compatible with the theorem and does not answer *для чего* (“what for”).
13. Historical origin is not required.
14. Self-application preserves the internal result and the external fence.

Not Established by This Record

- that formal checking is impossible;
- that arithmetic outputs are not reproducible;
- that a particular referent exists or is absent;
- which external truth assessment to choose;
- that an implementation works or does not work;
- that a named rocket flies or does not fly;
- that an action has or lacks a goal;
- that meaning exists or is absent;
- that contradiction is necessarily explosive;
- correctness of infinite, probabilistic, defeasible, or oracle systems;
- a theorem about every object named by the word *математика* (“mathematics”);
- the historical origin of mathematics;

- global novelty or priority for Phase 5.

This is not a concession to the external. It is a prohibition on extending an exact theorem by an unwritten quantifier.

23. The Final Boundary of Phase 5

A finite surface record may be fully decoded. An opaque code sequence may be admitted. Every non-axiom line may have a prior-line ground-set witness. The final accepted code may exactly match the target. The runtime may generate a complete trace.

All of this is preserved:

```
proof_status: ACCEPTED
terminal_witness.kind: TARGET-ACCEPTED
trace_scope_status: DECLARED-FINITE-TRACE-AND-CONTRACT-ONLY
```

But one surface receives ACCEPTED with R:MP enabled and REJECTED with R:MP disabled. Decoding, statement codes, axioms, complete rules, target, and payload do not change.

Therefore:

surface $\nRightarrow$ proof-status без read-contract .

The preserved source-form conclusion reads: “surface does not imply proof status without a read contract.”

And one formal result is compatible with different external completions. Therefore:

accepted derivation $\nRightarrow$ referent/truth/application/purpose/meaning без bridge .

The preserved source-form conclusion reads: “accepted derivation does not imply referent, truth, application, purpose, or meaning without a bridge.”

A rocket may satisfy a flight criterion. A calculation may be reproducible. The lines 100, 200, and 300 лет (“300 years”) may enter new records. None of these results places purpose inside the original lines.

Proof is not located in the written form.

It is a status of the written form under a contract.

And that status does not present what is proved outside the contract.

Source Status

This work continues the author's internal beforeword line and Phases 0–4. External comparative sources are not included among the premises. Historical origin, global novelty, and world priority are not established by this record.

01. THE PROOF-STATUS READSET THEOREM

A PROOF DOES NOT PRESENT WHAT IT PROVES

0. Scope of the result

This file specifies the finite exhibited family of Phase 5A records. It does not speak about everything ever named by the word `математика` (*mathematics*), and it introduces no historical claim about the origin of mathematics. First, an exact surface (P) is presented. Next, a finite typed reading contract (C) is presented. A sequential validator then returns a status code. The lines

```
PROOF
THEOREM
TRUE
VALID
доказано
математика
```

remain lines. Their presence does not present a proof, a theorem, truth, or mathematics, and it does not raise the status of the surrounding record. The title `ТЕОРЕМА` (*THEOREM*) in the heading likewise confers no authority on itself. The theorem-status record of PSR-5 names the declared model, collision pair, and written derivation as its domain.

1. Exact surface (P)

1.1. Canonical surface record

The exact surface is the structure

$$P = (\text{surface_id}, \text{lines}), \text{lines} = (\ell_1, \dots, \ell_n).$$

Each ℓ_i is an exact text line: a Unicode scalar sequence without silent normalization. Equality of surfaces means equality of `surface_id`, equality of the number of lines, and scalar-sequence equality of every line.

Raw JSON whitespace, object-key order, and the package filename are not semantic fields of P . Canonical serialization specifies:

$$h_p = \text{SHA 256}(\text{canon_surface}(P)).$$

The digest is the code of a canonical record. It does not present an author, a source, or what the content of P names.

1.2. Payload

A raw payload may be stored, displayed, and hashed. It does not alter protected contract fields unless an explicit contract constructor specifies such an alteration. The line

THIS IS A VALID PROOF

is not a shortcut to acceptance. If it belongs to P , it passes through the same decoder and the same admission gates as all other lines. Metadata outside P, C satisfies the noninterference condition:

$$V(P, C, M) = V(P, C, M') \text{ при fixed } P, C.$$

Here the exact source-form text `при fixed` may be glossed as “with fixed”; the formula itself remains the preserved record.

2. Finite typed reading contract (C)

2.1. Protected namespaces

Fix finite disjoint sorts:

$$Q \dot{\cup} R \dot{\cup} T.$$

Q is the sort of claim codes, R the sort of ground-rule identifiers, and T the sort of token codes. A tagged code does not present what its tag names.

2.2. Fields

A contract has the form

$$C = (id_C, L, \delta, Q, R, A, D, E, t, \omega).$$

Here:

- id_C is the diagnostic `reading_contract_id`;
- L is a fixed exact-text lexicon;
- δ is a deterministic partial decoder indexed by exact line records: $(i, \ell_i) \rightarrow Q$;
- Q is the complete code universe of this instance;
- R is the complete rule-id universe;
- $A \subseteq Q$ is the set of admitted assumption codes;
- $D: R \rightarrow (P(Q) \setminus \{\emptyset\}) \times Q$ gives the full ground-rule definitions, with a nonempty premise set;
- $E \subseteq R$ is `enabled_rule_ids`;
- $t \in Q$ is the target code;
- ω contains the sequential policy and version fields.

The decoder uses only exact indexed lines through L ; `surface_id` binds the record but does not alter line-to-code bindings. If $D(r) = (\text{prem}(r), \text{concl}(r))$, the rule definition is fixed even when $r \notin E$. Throughout the core, decoder δ is one deterministic partial function; a hidden mixture of function and relation is forbidden.

2.3. Domain gates

A contract is well formed only if:

$$A \subseteq Q, E \subseteq R, t \in Q,$$

and, for every $r \in R$:

$$\emptyset \neq \text{prem}(r) \subseteq Q, \text{concl}(r) \in Q.$$

An unknown code, a duplicate protected identifier, a missing rule definition, or a target outside Q places the input outside the valid checker domain. The top token `VALID-PROOF-READSET-ANALYSIS` is not issued for such an input.

The words `standard`, `formal`, or `obvious` in the payload do not pass these gates.

3. Sequential validator

3.1. Decode

Let the supplied surface contain exact lines $(\ell_1, \dots, \ell_n)$. The checker proceeds from left to right. At reached index i , it first computes $\delta_C(i, \ell_i)$. If the value is undefined, the contract-local fields are:

```
proof_status: UNPARSED
terminal_witness.kind: UNMAPPED-SURFACE-LINE
```

and the run stops immediately. If the value is defined, one code is fixed:

$$q_i = \delta_C(i, \ell_i) \in Q.$$

Traversal reaches the next index only if every reached line has been decoded and admitted. A later unmapped line therefore does not override an earlier `LINE-NOT-ADMITTED`.

3.2. Step admission

For every reached decoded $i \in \{1, \dots, n\}$:

$$H_i = \{q_j \mid 1 \leq j < i\}.$$

The exact condition is:

$$\text{Good}_i(C, P) \Leftrightarrow q_i \in A \vee \exists r \in E : \text{concl}(r) = q_i \wedge \text{prem}(r) \subseteq H_i.$$

Premises must occur earlier than the conclusion. A future line does not support an earlier step. If the least i fails GOOD, the contract-local fields are:

```
proof_status: REJECTED
terminal_witness.kind: LINE-NOT-ADMITTED
```

The diagnostic stores i , q_i , and the enabled rule ids considered.

3.3. Full admission predicate

$$ADM_C(P) : \Leftrightarrow \exists n > 0, \exists q_1, \dots, q_n : \bigwedge_{i=1}^n (q_i = \delta_C(i, \ell_i) \wedge Good_i(C, P)) \wedge q_n = t.$$

If every step is good but $q_n \neq t$, the contract-local fields are:

```
proof_status: REJECTED
terminal_witness.kind: TARGET-LINE-MISMATCH
```

If ADM holds, the contract-local fields are:

```
proof_status: ACCEPTED
terminal_witness.kind: TARGET-ACCEPTED
```

Here ACCEPTED is the value of a field named as a contract result, not a standalone claim. No field value is abbreviated to bare PROOF or bare TRUE.

3.4. Validator

Let j be the least reached index at which either $\delta_C(j, \ell_j)$ is undefined or the decoded q_j fails $Good_j$.

The exact sequential status is:

$$V_C(P) := proofStatus_C(P) = \begin{cases} UNPARSED, & j \text{ существует и } \delta_C(j, \ell_j) \text{ undefined,} \\ REJECTED, & j \text{ существует и } q_j \text{ decoded, но } \neg Good_j, \\ ACCEPTED, & j \text{ не существует и } n > 0 \text{ и } q_n = t, \\ REJECTED, & j \text{ не существует, но final target gate не выполнен.} \end{cases}$$

The preserved source-form clauses read, respectively: “ j exists and the decode is undefined”; “ j exists and q_j is decoded, but GOOD fails”; “ j does not exist, $n > 0$, and $q_n = t$ ”; and “ j does not exist, but the final target gate is not satisfied.”

The field $\tau_C(P)$, written at the runtime path `terminal_witness.kind`, is stored separately as a diagnostic of the reason or terminal gate. It is not a second component of $V_C(P)$, and it does not participate in the classification of PSR-5 divergence. $V_C(P)$ is the code of a relation between P and C . It does not present the referents of claim codes.

4. Extensional ReadSet

4.1. Sufficient set

Let an input record have coordinates $x = (x_1, \dots, x_m)$. A set $S \subseteq \{1, \dots, m\}$ is sufficient for F if:

$$res_S(x) = res_S(y) \Rightarrow F(x) = F(y).$$

4.2. Necessary coordinate

Coordinate k is extensionally necessary if there exists a collision pair x, y :

$$res_{\neg k}(x) = res_{\neg k}(y), x_k \neq y_k, F(x) \neq F(y).$$

Every sufficient ReadSet must distinguish such a pair. Here ReadSet is not a psychological claim that the checker “sees” a field; it is extensional necessity specified by equality and an output collision.

4.3. Contract projection

Let k_C be the index of the contract projection in the input tuple, with $proj_C(x) = C$. The general PSR-5 asserts the necessity of k_C , not membership of the record C itself in a set of indices.

For the stronger minimal pair, introduce coordinate

$$k_E = index(E), proj_E(C) = E.$$

In the runtime schema, coordinate E is written as the field `enabled_rule_ids`.

Semantic erasure removes the diagnostic id and the enabled field:

$$erase_{id,E}(C).$$

It is this projection that must agree in the minimal pair; only E differs.

`enabled_rule_ids`

All other semantic coordinates are fixed extensionally. The diagnostic `reading_contract_id` may differ, but it is not part of the semantic validator projection.

5. PSR-5

5.1. General statement

Let exact P be fixed and let C_1, C_2 belong to the declared finite contract family C . If:

$$V_{C_1}(P) \neq V_{C_2}(P),$$

then the contract projection that distinguishes C_1, C_2 is necessary for status on C .

Here and in PSR-5, $V_C(P)$ is only the value of the `proof_status` field. PSR-PREMISE is therefore true exactly when `proof_status` differs; the separate values $\tau_{C_1}(P), \tau_{C_2}(P)$ are not compared by the divergence classifier.

$$V_{C_1}(P) \neq V_{C_2}(P) \Rightarrow k_C \in \text{ReadSet}_C(V)$$

The box denotes the extensional necessity of proj_C , not the necessity of every unused bit. For the minimal pair, the stronger form is:

$$k_E \in \text{ReadSet}_{\{C_1, C_2\}}(V).$$

5.2. Equivalent impossibility

There is no core surface-only function $F: P \rightarrow S_V$ that, for one P and without receiving the contract difference, returns both distinct required statuses:

$$\nexists F: F(P) = V_{C_1}(P) \wedge F(P) = V_{C_2}(P).$$

5.3. Proof

Suppose such an F exists. Its argument is exact P in both instances, so its output $F(P)$ is one value. But PSR-PREMISE requires two different outputs. Contradiction. If a sufficient ReadSet does not distinguish C_1, C_2 , the pair agrees on the read coordinates while the outputs differ, violating SUFFICIENT. The distinguishing contract projection is therefore necessary. $\square$

5.4. Quantifier scope

PSR-5 quantifies over the exhibited C , not over *всей математике* (*all of mathematics*). The phrase *вся математика* (*all mathematics*) does not specify a domain merely by appearing. The result is strong enough without a historical-origin claim: in the declared family, proof status is not presently a function of the lines alone.

6. Minimal collision pair

6.1. Fixed surface

$$P_{ab} = (\text{surface} - \text{psr} - \text{minimal}, (a, a \rightarrow b, b)).$$

Readable rendering:

```
a
a → b
b
```

The exact canonical $\text{surface_sha256 } h_{ab}$ is the same in both instances; raw container JSON bytes are not compared.

6.2. Fixed coordinates

The following agree in both semantic contracts:

- the exact line lexicon;
- the deterministic parser;
- code universe $Q = \{A, I_{AB}, B\}$;
- rule universe $R = \{m p\}$;
- assumptions $A_0 = \{A, I_{AB}\}$;
- the full ground-rule definition;
- target $t = B$;
- the sequential policy;
- inert payload;
- validator version.

The diagnostic $\text{reading_contract_id}$ may differ only as an ignored label. After erasing id_C and the enabled field:

$$\text{erase}_{id,E}(C_1) = \text{erase}_{id,E}(C_2).$$

The per-line decoder is fixed; the tuple below is shorthand for three indexed applications:

$$(\delta(1, \ell_1), \delta(2, \ell_2), \delta(3, \ell_3)) = (A, I_{AB}, B).$$

The ground-rule definition is fixed:

$$D(m p) = (\{A, I_{AB}\}, B).$$

6.3. Sole difference

$$E_1 = \{m p\}, E_2 = \emptyset.$$

There is exactly one semantic difference:

$$E_1 \triangle E_2 = \{m p\}.$$

That is, enabled_rule_ids differs by one id; diagnostic contract ids are excluded from validator semantics.

6.4. First output

In C_1 :

$$Good_1=1, Good_2=1, Good_3=1$$

because $m p \in E_1$, its conclusion is B , and its premises occur in H_3 . The result is:

```
proof_status: ACCEPTED
terminal_witness.kind: TARGET-ACCEPTED
```

6.5. Second output

In C_2 , the first two steps are assumptions, but:

$$Good_3=0$$

because $E_2 = \emptyset$. The result is:

```
proof_status: REJECTED
terminal_witness.kind: LINE-NOT-ADMITTED
```

6.6. Strengthened conclusion

Not only P , but also the lexicon, parser, codes, assumptions, full rule definitions, target, payload, and validator are fixed. The output collision is induced by the single enabled-set bit. Thus `enabled_rule_ids` is a necessary coordinate for this exhibited family. A rule definition that is written but disabled does not become enabled by its text.

7. ADM-5

7.1. Admission inputs

GOOD displays the exact sources of admission:

- assumption membership $q_i \in A$;
- enabled-rule membership $r \in E$;
- fixed conclusion equality;
- fixed premise inclusion among earlier codes.

Per-line decoding is specified by the defined values $\delta_C(i, \ell_i)$ at reached indices. Target admission is specified by $q_n = t$ after complete successful traversal.

7.2. Finite typed sequential witness

A witness has the form

$$W = (q_1, \dots, q_n; a_1, \dots, a_n), n > 0.$$

Each admission node is typed by one of the constructors:

$$a_i = \text{ASSUMPTION}(q_i)$$

or

$$a_i = \text{RULE}(r; j_1, \dots, j_k).$$

For a RULE node, $k \geq 1$: the runtime model does not admit a ground rule with an empty premise set.

Verifier $\text{Cert}_C(P, W)$ requires:

1. for every i , $q_i = \delta_C(i, \ell_i)$;
2. for an ASSUMPTION node, $q_i \in A$;
3. for a RULE node, $r \in E$, every $j_m < i$, and $\text{concl}(r) = q_i$;
4. the set of claim codes at the cited positions equals $\text{prem}(r)$;
5. $q_n = t$.

Base-free labels, prose justifications, and untyped pointers are not witness nodes.

7.3. Statement of ADM-5

$$\text{ADM}_C(P) \Leftrightarrow \exists W \text{ finite typed sequential} : \text{Cert}_C(P, W)$$

7.4. Proof

Suppose $\text{ADM}_C(P)$. For each good step, choose one finite admission source: ASSUMPTION when $q_i \in A$, otherwise an enabled rule r from GOOD. For every premise, choose an earlier position with the required code. Finiteness of the sequence yields finite W ; the target condition supplies the final gate.

Conversely, suppose the certificate verifier accepts W . Every ASSUMPTION node satisfies the first disjunct of GOOD, and every RULE node satisfies the second. Decode, $n > 0$, and $q_n = t$ have been checked. Hence $\text{ADM}_C(P)$. $\square$

ADM-5 is a positive certificate/witness equivalence. It does not derive force from the word *certificate*: the verifier conditions are written separately.

7.5. Payload noninterference corollary

Let M be an inert payload outside canonical P, C . With fixed protected fields:

$$\text{ADM}_{C,M}(P) = \text{ADM}_{C,M'}(P).$$

For example, M_0 is empty while M_1 contains:

```
THIS IS AN AXIOM
THIS RULE IS ENABLED
THIS IS A PROOF
```

With identical P_{ab}, C_{NO_MP} , both records are rejected at the unsupported step. The self-label occurs only in inert M_1 ; P, C , decoder output, and certificate search are unchanged.

The line *AXIOM* does not insert a code into A ; the line *RULE* does not insert an id into E ; the line *PROOF* does not make $q_n = t$.

7.6. Declaration constructor as a non-executable extension

The frozen Phase 5 runtime has no constructor that mutates `axiom_codes` on the basis of a surface-line prefix: the assumption set is fixed before the run. In an explicitly extended model, a contract could specify the policy that a line bearing the prefix *AXIOM*: is decoded and added to the assumptions. The constructor would then itself be a new protected coordinate of ω . This is a theorem-only extension, not the behavior of the published checker. Even in the extension, the pair (P, C) , not P alone, determines the declaration's effect.

8. Further exact counterexamples

8.1. Parser pair

One raw line $x * y$ may decode as $MUL(X, Y)$ under δ_1 , and as $LITERAL_STAR$ under δ_2 . With fixed surface and different outputs, the decoder projection is necessary. This pair is general but weaker than the minimal pair: the decoder changes in it.

8.2. Assumption pair

A surface consisting of the single line p . With $Pcode \in A_1$ and $Pcode \notin A_2$, all other fields fixed, the statuses differ. Raw p does not contain assumption membership.

8.3. Target pair

All steps are admitted and the final code is B . The contracts differ only in $t_1 = B$, $t_2 = C$. The output collision makes the target coordinate necessary. The final line does not select a target autonomously.

8.4. Order control

The surfaces $a / a \rightarrow b / b$ and $b / a / a \rightarrow b$ are different. This is surface sensitivity, not a PSR collision at fixed P . The control is needed so that a change in the exact line sequence is not attributed to the contract theorem.

9. Metaproof

9.1. Lift

Let M be named a metaproof of admission for contract C . Verification requires a metacontract D :

$$W_D(M, C).$$

D specifies a metaparser, meta-assumptions, metarules, and a metatarget.

9.2. Meta collision

If, with fixed (M, C) :

$$W_{D_1}(M, C) \neq W_{D_2}(M, C),$$

then D is necessary by the same collision argument. A metaproof moves the question to a new contract. It does not create contract-free admission.

9.3. Finite stopping point

No claim of a mandatory infinite regress is made. A procedure may stop at explicitly declared D . But the boundary `принимаем D` (we accept D) remains an input. The line `D validates itself` does not remove D from W_D .

10. Gates

Gate 0 — EXACT-SURFACE

- `surface_id` and exact lines are presented;
- there is no silent scalar normalization;
- `surface_sha256` is written;
- the collision pair preserves P .

Gate 1 — CONTRACT-SHAPE

- finite typed Q, R ;
- total ground definitions D on R , with $\emptyset \neq \text{prem}(r) \subseteq Q$;
- $A \subseteq Q, E \subseteq R, t \in Q$;
- no unknown protected ids.

Gate 2 — DETERMINISTIC-DECODE

- the exact-line lexicon is fixed;
- δ is a deterministic partial function indexed by exact line records;
- every reached supplied line is accounted for, including a reached line with `line_text == ""`;
- the first reached undefined decode receives `proof_status: UNPARSED`; an earlier terminal rejection prevails over every later line.

Gate 3 — SEQUENTIAL-GOOD

- every q_i satisfies GOOD;
- the rule id is enabled;

- premises occur earlier;
- a self-label is not a shortcut.

Gate 4 — TARGET

- $n > 0$;
- $q_n = t$;
- the target belongs to the declared universe.

Gate 5 — PROVENANCE

- artifact_sha256, surface_sha256, and per-contract derivation_core_sha256 are written;
- the admission kind of every step is written;
- the enabled id and premise indices are recorded;
- the validator version is fixed.

Failure at a gate excludes acceptance only under the named contract. It creates neither a bare negative nor a world status.

11. Certificate

Executable output uses only the frozen field vocabulary:

- top status = VALID-PROOF-READSET-ANALYSIS;
- artifact_sha256 — canonical parsed artifact;
- surface_sha256 — **protected** {surface_id, lines};
- per-contract reading_contract_id;
- proof_status = ACCEPTED, REJECTED, OR UNPARSED;
- derivation_core_sha256;
- accepted_prefix_length;
- trace;
- terminal_witness;
- pair token READING-CONTRACT-REQUIRED-FOR-PROOF-STATUS;
- readset_delta_fields.

Terminal-kind vocabulary:

TARGET-ACCEPTED
 LINE-NOT-ADMITTED
 TARGET-LINE-MISMATCH
 UNMAPPED-SURFACE-LINE

Reader gloss, not machine tokens: `ACCEPTED` means ADM true under named contract; `REJECTED` means that the first reached mapped line failed admission or that complete traversal failed the target gate; `UNPARSED` means that the first reached line lacked an exact mapping under the named contract.

Replay is exact only if:

$$h_A' = h_A, h_S' = h_S, h_D' = h_D.$$

Here h_A, h_S, h_D denote, respectively, the runtime fields `artifact_sha256`, `surface_sha256`, and `derivation_core_sha256`.

The trace and terminal witness must also match. The certificate displays a recorded dependency; it does not present the referent of a claim code.

This is a replay requirement, not the PSR classification rule: pairwise PSR divergence is computed only from inequality of `proof_status`; `terminal_witness.kind` remains a separate diagnostic coordinate.

12. Forbidden exports

From

$$V_C(P) = \text{ACCEPTED}, \tau_C(P) = \text{TARGET} - \text{ACCEPTED}.$$

the following do not follow without new coordinates:

world truth, fact, referent, application, goal, meaning.

The following abbreviations are forbidden:

```
accepted here = accepted everywhere
familiar rule = rule enabled
machine checked = contract free
meta checked = autonomous authority
```

The next file, Phase 5B, introduces a separate export binding *I* rather than concealing it in the word `proof`.

13. Extensions

13.1. Nondeterministic output

If the validator returns a set $V_C(P) \subseteq S_V$, PSR applies to different output sets. The contract coordinate remains necessary.

13.2. Probabilistic output

If the output is a distribution $\mu_{C,P}$, then $\mu_{C_1,P} \neq \mu_{C_2,P}$ creates a collision. Probability does not erase the ReadSet.

13.3. Interactive run

Transcript T enters the input $V_C(P, T)$. PSR requires fixed (P, T) ; changing T produces a different pair.

13.4. Executable package

The kernel, imports, environment, and flags belong to an extended C^+ . Status remains ...UNDER-DECLARED-CONTRACT. The word `assistant` does not present an external checker.

14. Limits

PSR-5 establishes the necessity of a coordinate in the exhibited family. It does not rank contracts and does not assert that accepted derivations are useless. It does not assert that computation cannot alter a device. An altered device would be a separate presented record, but not automatically the referent of a word. It does not establish the historical origin of mathematics, a person's state, universal impossibility of binding, or world falsity. ADM-5 does not forbid a declaration constructor. It requires the constructor to be displayed in protected C . The title `proof`, the PSR-5 formula, and the sign $\square$ do not acquire authority from themselves.

15. Falsifiers

15.1. PSR falsifier

To defeat the PSR collision, present an $F(P)$ that, for one exact P and without receiving the distinction between C_1, C_2 , returns both distinct deterministic statuses in one output namespace. If F returns a pair or a set, the task has changed: contract-indexed statuses have been packaged, not obtained from P alone.

15.2. Pair defeaters

The application fails if:

- the surfaces differ;
- the outputs agree;
- a coordinate other than the declared one changes;
- one contract lies outside declared C ;
- different status projections are compared;
- the contract is hidden inside the object named P .

15.3. ADM falsifier

If only untrusted metadata changes the output while P, C are fixed, the checker violates PAYLOAD-NONINTERFERENCE. This detects a hidden input, not metadata's self-authority.

16. Self-boundary

This Markdown file is an exact surface P_{PSR} . The lines PSR-5, теорема (*theorem*), and $\square$ belong to it. Validation requires C_{PSR} : a Markdown/math parser, an assumption set, ground rules, enabled ids, and a target. Another contract may return another scoped status. This is not an exception; it is PSR-5's own instance. The file does not present an author, a reader, mathematics, truth, or the world. It presents lines and declared relations among codes.

17. Final Phase 5A cut

Lines are written first. To obtain a derivation status, a parser, assumptions, ground rules, enabled ids, and a target are introduced separately. The minimal pair fixes everything, including the full rule definitions and payload. Only

enabled_rule_ids

changes, and acceptance changes with it.

$$(P, C) \mapsto V_C(P), P \mapsto \text{о} \text{ба status alone.}$$

The preserved source-form formula says that P alone does not map to both statuses.

A self-label does not replace C . A metaproof introduces D ; it does not erase the contract coordinate.

Contract-local proof_status: ACCEPTED records the status of passing the declared gates. The separate diagnostic terminal_witness.kind: TARGET-ACCEPTED records the terminal gate and does not classify PSR-5 divergence. Neither field presents what is proved, a referent, truth, a source, an application, a goal, or meaning.

Top analysis token:

VALID-PROOF-READSET-ANALYSIS

Pair result:

READING-CONTRACT-REQUIRED-FOR-PROOF-STATUS

Fences:

NOT-MACHINE-VERIFIED
DECLARED-FINITE-TRACE-AND-CONTRACT-ONLY
NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER
NO-HISTORICAL-ORIGIN-ATTESTATION

The final line, too, remains a line.

Exact source-form coordinates. The following forms are retained as written objects from the Russian source. Their inclusion records a reading-map coordinate; it neither substitutes them into the English prose nor exports anything the forms name.

Inline mathematical forms:

$P; C; C.$

02. THE NO-AUTONOMOUS-EXPORT THEOREM

AN ACCEPTED DERIVATION DOES NOT PRESENT AN APPLICATION

0. Domain of the result

This file continues Phase 5A after the scoped status:

```
proof_status: ACCEPTED
terminal_witness.kind: TARGET-ACCEPTED
```

A fixed derivation core

$$P, C$$

is admitted. Phase 5B asks whether an accepted core is sufficient to obtain an application code, goal code, or external-claim code autonomously. The NEX-5 answer concerns a declared finite binding family. It does not quantify over *всем применениям* (“all applications”) and establishes no world-level negative thesis. The strings

```
ROCKET
MARS
PURPOSE
TRUE IN THE WORLD
APPLICATION
```

remain strings. They present no rocket, Mars, purpose, world, or application.

1. Fixed accepted derivation core

1.1. Core

Let $P = (\text{surface_id}, \text{lines})$ be an exact canonical surface record and C a finite typed Phase 5A reading contract. Assume:

$$ADM_C(P) = 1$$

and the contract-local fields:

```
proof_status: ACCEPTED
terminal_witness.kind: TARGET-ACCEPTED
```

The selected target code is:

$$t = t_C \in Q_C.$$

Acceptance remains a relation

$$P, C$$

. It does not turn t into a referent.

1.2. Derivation-core serialization

The canonical derivation core contains exactly:

- `surface_sha256` of canonical `{surface_id,lines}`;
- complete ordered decoding records;
- `axiom_codes`;
- the complete normalized `rules[{{rule_id,premise_codes,conclusion_code}}` catalog, without per-rule payload;
- `enabled_rule_ids`;
- `target_code`.

The checker constructs the named JSON object—not a tuple or array:

```
{
  "surface_sha256": "...",
  "decoding": [],
  "axiom_codes": [],
  "rules": [],
  "enabled_rule_ids": [],
  "target_code": ""
}
```

and defines:

$$d = \text{SHA } 256(\text{canon}(D)).$$

where D is that named object. Object-key order in the source serialization is not a coordinate: the canonicalizer supplies canonical key order. Member order inside protected arrays remains part of the object value. The runtime field is:

```
derivation_core_sha256
```

1.3. What the core digest excludes

d excludes:

- export binding I ;
- outer descriptions;
- application labels;
- goal labels;
- claim labels;
- package filename;
- untrusted metadata;
- trace and terminal witness;

- diagnostic reading_contract_id;
- raw container JSON bytes.

This exclusion is mandatory for the collision pair: if I were included in the digest being compared, the fixed-core premise would be hiddenly violated. Full artifact hashes under $I_1 \neq I_2$ are expected to differ. NEX-5 compares equal derivation_core_sha256, not equal full artifact hashes. A digest remains code bytes. Its name presents neither a derivation, a referent, nor a source.

2. Opaque export sorts

2.1. Separate namespaces

Fix finite tagged namespaces:

$$Q \dot{\cup} U \dot{\cup} G \dot{\cup} W.$$

Here Q is internal claim-codes; U , application-codes; G , goal-codes; and W , external-claim-codes. external means only a separate namespace. It presents no outside.

2.2. Opaque-code discipline

Normative examples use:

```
APP-07
GOAL-03
CLAIM-11
```

Readable labels belong to optional outer descriptions, not to code identity. Even `rocket` is not a rocket. Even `GOAL` is not a goal.

2.3. No cross-sort coercion

Without an explicit constructor, the conversions

$$q \in Q \not\Rightarrow q \in U,$$

$$u \in U \not\Rightarrow u \in G,$$

$$g \in G \not\Rightarrow g \in W.$$

are prohibited. The same character sequence in different tagged namespaces does not remove the type boundary.

3. Export binding I

3.1. Typed record

The binding record is:

$$I = (U, G, W, B_I, v_I).$$

U, G, W are declared finite universes; v_I is the canonicalization and version policy. B_I is a finite partial map:

$$B_I: D \times Q \rightarrow U \times G \times W,$$

where D is the namespace of derivation-core digests. Lookup uses the exact pair (d, t) .

3.2. Binding entry

An entry

$$B_I(d, t) = (u, g, w)$$

establishes only a recorded association among four codes. It does not establish that u has a referent; that g is anyone's goal; that w is true; that a named application was performed; that the core causally created an external object; or that the words of an outer description correspond to anything beyond the record.

3.3. Well-formedness

I is well-formed only if all digest keys are canonical; all internal codes belong to Q ; all outputs are typed in $U \times G \times W$; a duplicate key has exactly one output; there is no implicit default binding; and there is no wildcard outside an explicit policy. In the theorem-only model, violation receives the exposition diagnosis:

INVALID-EXPORT-BINDING-RECORD

This is not a token in the executable checker verdict schema. `official` in payload does not pass well-formedness.

4. Abstract export function

Sections 2–10 define a theorem-only finite model. `BOUND-EXPORT-UNDER-DECLARED-BINDING`, `UNBOUND-EXPORT-UNDER-DECLARED-BINDING`, and `INELIGIBLE-CORE-FOR-EXPORT-UNDER-DECLARED-CONTRACT` are abstract values of X_I , not machine tokens of the executable checker.

4.1. Lookup

For a fixed accepted core (d, t) :

$$\ell_B := \text{BOUND} - \text{EXPORT} - \text{UNDER} - \text{DECLARED} - \text{BINDING}.$$

$$\ell_U := \text{UNBOUND} - \text{EXPORT} - \text{UNDER} - \text{DECLARED} - \text{BINDING}.$$

$$X_I(d, t) = \begin{cases} (\ell_B, u, g, w), & B_I(d, t) = (u, g, w), \\ (\ell_U, \perp, \perp, \perp), & (d, t) \notin \text{dom}(B_I). \end{cases}$$

(EXPORT-VALIDATOR)

UNBOUND... means lookup nonmembership only under exact I . It is not shortened to NO APPLICATION, NO GOAL, FALSE, OR NO REFERENT.

4.2. Core acceptance gate

If $ADM_C P = 0$, the abstract function does not perform lookup as export of an accepted derivation and returns:

INELIGIBLE-CORE-FOR-EXPORT-UNDER-DECLARED-CONTRACT

This is a status pipeline, not a world claim.

4.3. Determinism

In the normative core, B_I is a deterministic partial function. A multi-binding relation is treated separately in extensions. Function and relation must not be mixed in one theorem instance.

5. Extensional export ReadSet

5.1. Collision definition

Let k_I index the binding coordinate and $\text{proj}_I(O) = I$. Coordinate k_I is necessary for (X) on a family I if some $I_1, I_2 \in I$ satisfy:

$$d_1 = d_2 = d, t_1 = t_2 = t,$$

$$I_1 \neq I_2, X_{I_1}(d, t) \neq X_{I_2}(d, t).$$

This is an extensional definition. It does not assert that a program psychologically “reads” a binding.

5.2. Outer records

Let the full outer artifact be:

$$O_i = (d, t, I_i, Z_i),$$

where Z_i consists of opaque descriptions and metadata. A collision pair requires equality of derivation core:

$$\text{core}(O_1) = \text{core}(O_2) = (d, t).$$

Full artifact equality is neither required nor normally true because $I_1 \neq I_2$. Descriptions Z_i do not enter binding lookup unless an explicit policy makes them protected input.

6. NEX-5

6.1. Statement

Let (P, C) be accepted under a declared contract, d a fixed core digest, t a fixed target, and $I_1, I_2 \in I$.
If

$$X_{I_1}(d, t) \neq X_{I_2}(d, t),$$

then coordinate k_I , whose projection is proj_I , is necessary for the export status/value on declared family I :

$$X_{I_1}(d, t) \neq X_{I_2}(d, t) \Rightarrow k_I \in \text{ReadSet}_I(X)$$

Therefore accepted (P, C) alone does not determine both export outputs.

6.2. Impossibility of a core-only function

$$\nexists F : F(d, t) = X_{I_1}(d, t) \wedge F(d, t) = X_{I_2}(d, t)$$

when the required outputs are distinct and neither I nor an encoded equivalent is among the arguments.

6.3. Proof

Assume a core-only function $F(d, t)$. The collision pair has the same d, t , so the function has one output. NEX-PREMISE requires different outputs. Contradiction. A ReadSet that does not distinguish I_1, I_2 retains equal coordinates while X differs, so it is not sufficient. The distinguishing binding projection is therefore necessary. $\square$

6.4. Scope

NEX-5 concerns the exhibited I . It does not assert that no application is possible. It states exactly that different exports under a fixed accepted core require a coordinate recording the difference. реальное применение (“real application”) does not expand the domain or present an application.

7. Minimal binding pair

7.1. Fixed core

Take the accepted P_{ab}, C_{MP} of Phase 5A:

```
a
a → b
b
```

All derivation-protected fields are fixed. Target $t = B$; core digest d_{ab} is fixed.

7.2. Fixed universes

$$U = \{APP-07, APP-08\},$$

$$G = \{GOAL-03, GOAL-04\},$$

$$W = \{CLAIM-11, CLAIM-12\}.$$

The universes, canonicalization, and version policy are fixed.

7.3. Binding I_1

$$B_{I_1}(d_{ab}, B) = (APP-07, GOAL-03, CLAIM-11).$$

Output:

```
BOUND-EXPORT-UNDER-DECLARED-BINDING
APP-07
GOAL-03
CLAIM-11
```

7.4. Binding I_2

$$B_{I_2}(d_{ab}, B) = (APP-08, GOAL-04, CLAIM-12).$$

Output:

```
BOUND-EXPORT-UNDER-DECLARED-BINDING
APP-08
GOAL-04
CLAIM-12
```

7.5. Collision

The surface, contract, admission certificate, target, and derivation-core digest are equal. Only the binding entries differ. The export values differ. By NEX-5, I is necessary. Neither (P) nor scoped acceptance selects APP-07 rather than APP-08.

7.6. Outer-description control

Z_1 may contain ракета, Марс, and цель полёта—“rocket,” “Mars,” and “flight goal.” Z_2 may contain other labels. These strings do not participate in the minimal collision proof and present none of what they name.

8. Unbound is not a negative world status

8.1. Empty binding

Let:

$$B_{I_0} = \emptyset.$$

Then:

$$X_{I_0}(d, t) = (\text{UNBOUND} - \text{EXPORT} - \text{UNDER} - \text{DECLARED} - \text{BINDING}, \perp, \perp, \perp).$$

8.2. Prohibited transition

The exact formula is retained:

$$(d, t) \notin \text{dom}(B_I) \neq \text{названное отсутствует}.$$

The Russian formula text reads “what is named is absent.” The following are likewise prohibited:

```
UNBOUND  $\Rightarrow$  FALSE
UNBOUND  $\Rightarrow$  NO-REFERENT
UNBOUND  $\Rightarrow$  NO-GOAL-IN-WORLD
```

8.3. Reason

Lookup establishes only nonmembership of a key in finite map B_I . The model contains no world-state predicate. A negative status in another domain requires a separate finite contract, provenance, and completeness boundary. Silence in a binding record creates none of them.

8.4. Missing I

If no binding record is presented, executable fields are:

```
export_binding_count: 0
export_results: []
binding_coordinate_pairwise: []
```

The empty arrays report only zero explicit binding records in the input package. They report no absence of an application, goal, or referent.

9. A recorded binding does not present a referent

9.1. Record of association

Even a well-formed entry

$$B_I(d, t) = (u, g, w)$$

presents codes and a membership relation. It presents no object, action, or state described by outer labels.

9.2. Binding of a binding

Binding u to a new referent code r introduces another record:

$$J(u) = r.$$

But J again presents a recorded relation between codes. `referent-code` does not present a referent.

9.3. No magic terminal tag

The chain

$$t \xrightarrow{I} u \xrightarrow{J} r$$

may be a useful recorded chain. Neither its final tag nor its length creates an unrecorded presentation. `REAL OBJECT` remains a string.

9.4. Observed-outcome record

An exact trace K , labeled `наблюдаемым результатом` (“observed outcome”), and the link

$$L(u) = h_K.$$

add another surface and hash to the package. The label `observed` gives it no world truth.

10. Exact counterexamples

10.1. Same core, different binding

The minimal pair in § 7 directly counters:

accepted derivation uniquely determines export

One core has two exports under two bindings.

10.2. Same binding, different description

Let I be fixed and $Z_1 \neq Z_2$. If Z is untrusted, $X_I(d, t)$ is equal. Persuasive description does not change protected export.

10.3. Description without binding

Outer payload:

This proof applies to a rocket and proves its purpose.

with $B_I = \emptyset$ receives scoped UNBOUND.... The sentence synthesizes no entry.

10.4. Binding self-label

THIS IS THE ONLY TRUE APPLICATION

is not typed $B_I(d, t) = (u, g, w)$ without a declared parser and constructor.

10.5. Full-hash false control

Full artifact hashes $H(O_1)$ and $H(O_2)$ differ. It does not follow that the derivation cores differ. The correct equality witness is:

$$derivation_core_sha256(O_1) = derivation_core_sha256(O_2).$$

11. Export gates

Gate 0 — ACCEPTED-CORE

- (P,C) well-formed;
- ADM true;
- scoped acceptance present;
- target t exact.

Gate 1 — CORE-IDENTITY

- canonical core serialization fixed;
- derivation-core digest recomputed;
- binding and outer descriptions excluded;
- collision pair uses the same (d,t).

Gate 2 — BINDING-SHAPE

- I finite and typed;
- universes explicit;
- B_I a deterministic partial function;
- no conflicting duplicate key;
- no implicit default.

Gate 3 — EXACT-LOOKUP

- key exactly (d, t) ;
- output codes in $U \times G \times W$;
- unbound returns a scoped status;
- payload cannot synthesize an entry.

Gate 4 — POLARITY

- absence of an entry is only map nonmembership;
- no default FALSE;
- no default NO REFERENT;
- no default NO GOAL.

Gate 5 — PROVENANCE

- core digest recorded;
- binding digest recorded;
- matched key and entry recorded;
- policy/version recorded;
- outer descriptions separately hashed.

Failure of a gate blocks only the corresponding scoped export. It establishes no world status.

12. Executable specialization

Abstract B_I is not claimed to be a runtime map. The executable checker accepts explicit co-occurrence binding records and directly compares coordinate pairs.

12.1. Export results

Top-level field:

```
export_binding_count
```

Each `export_results[]` member contains:

```
binding_id
reading_contract_id
source_proof_status
source_derivation_core_sha256
source_target_code
claim_code
application_code
goal_code
export_status
export_status_sha256
```

Frozen `export_status` values:

```
EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE
EXPLICIT-BINDING-RECORD-WITHOUT-ACCEPTED-TRACE
```

The first establishes co-occurrence of an explicit binding record with an accepted trace under a named reading contract. It establishes no referent, successful application, adoption of a goal, or world truth.

12.2. Pairwise coordinate result

Each `binding_coordinate_pairwise[]` member contains:

```
left_binding_id
right_binding_id
reading_contract_id
left_export_status_sha256
right_export_status_sha256
comparison_status
```

Frozen `comparison_status` values:

```
BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS
NO-EXPORT-STATUS-DIVERGENCE-IN-THIS-PAIR
```

Classification is recomputed by direct comparison of `(source_target_code, claim_code, application_code, goal_code)`.

The first token specializes the NEX-5 collision argument only when both members additionally have `source_proof_status == ACCEPTED`, as the frozen positive binding fixtures do. Without that gate it remains a coordinate-comparison record. The second token reports equality of those tuples in the pair.

`left_export_status_sha256` and `right_export_status_sha256` are evidence fields, not the classification selector. Each digest covers exactly `reading_contract_id`, `source_proof_status`, `source_derivation_core_sha256`, `source_target_code`, `claim_code`, `application_code`, and `goal_code`; it excludes `binding_id` and the literal `export_status`.

12.3. Empty input and fence

When no binding records are presented:

```
export_binding_count: 0
export_results: []
binding_coordinate_pairwise: []
```

The arrays create no negative application/world status. The executable fence is:

```
NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER
```

Outer descriptions remain opaque payload. A binding record remains recorded co-occurrence, not presentation of a referent.

13. Relation to an operating device

13.1. Additional record

Suppose the package contains a trace K described by *ракета летит* (“the rocket is flying”). The trace may contain bytes, measurements, images, or event codes. All are additional presented records. Their presence changes neither (P,C) nor creates I without a protected constructor.

13.2. Application success

A new contract C_{succ} , predicate $\text{Succ}_{C_{succ}}(K)$, and scoped status may be defined:

SUCCESS-UNDER-DECLARED-APPLICATION-CONTRACT

This is a new readset (K, C_{succ}) . It does not become bare SUCCESS.

13.3. What for

A goal code g may be bound to application code u . The binding establishes only a code relation. *для чего?* (“what for?”) enters as a written question; an answer code enters as a written answer. Neither gains meaning from grammatical form. Even success under C_{succ} derives no goal or meaning without a declared binding.

14. Extensions

14.1. Multi-binding relation

If B_I is a relation, the output is a finite set:

$$X_I(d, t) \subseteq U \times G \times W.$$

Different sets under a fixed core again make I necessary.

14.2. Prioritized binding

If several entries are permitted, priority order belongs to (I). A change of output under changed order creates a separate necessary coordinate.

14.3. Dynamic registry

Timestamp, registry snapshot, and resolver version belong to expanded I^+ . *current* does not fix a snapshot.

14.4. Probabilistic export

For distribution $\mu_{I,d,t}$, different distributions under a fixed core give a NEX collision. Probability presents no referent.

14.5. Human selection record

Selection by an annotator is represented by codes (a, η) . They belong to the binding readset if they affect output. The model asserts no state of a named person.

15. Limits

NEX-5 does not assert that application is impossible; devices do not work; goals cannot be recorded; bindings are useless; all bindings are equally admissible; no code has a referent; the world is absent; mathematics historically arose from language; or proof systems compute nothing.

NEX-5 asserts necessity of (I) on the exhibited collision family and prohibits hidden export from an accepted core alone. recorded binding не предъявляет referent ("a recorded binding does not present a referent") is a boundary of the presented form: the shown entry consists of lines and codes. Its tag gives it no referent certification.

16. Falsifiers

16.1. NEX falsifier

Present a core-only $F(d, t)$ that, for fixed (d, t) , without receiving I_1/I_2 or an encoded equivalent, returns both distinct required deterministic outputs in one namespace. A pair/set output changes the codomain and packages alternatives; it does not select both as one value.

16.2. Collision defeaters

NEX does not apply if derivation-core digests differ; the target differs; core acceptance differs; outputs are equal; the binding difference is hidden in (d) ; one I_i lies outside the declared family; or a full artifact hash is mistakenly used instead of the core digest.

16.3. Negative-default falsifier

Deriving a negative world status from unbound lookup requires an explicit world-status contract and completeness certificate. That is expanded input, not a conclusion from missing (I).

16.4. Referent-presentation test

If an entry is claimed to present a referent, the exact record must show where what is presented is not another code or description, but what is named. Adding a label extends the record; it does not remove the boundary.

17. Self-boundary

This file is an exact surface P_{NEX} . It contains export, application, goal, referent, and NEX-5; none presents what it names. Formulas require a reading contract, and theorem status requires a declared metatheory. If a

website links the file under исследование (“research”), the link and label form an outer record; they create no authority. NEX-5 applies to itself: different outer bindings of the file under a fixed core require coordinate (I). Self-application does not expand the quantifier beyond the declared family.

18. Final Phase 5B cut

An accepted derivation core fixes:

$$FormalStatus(P, C) = (ACCEPTED, TARGET - ACCEPTED).$$

Export has another form:

$$(d, t, I) \mapsto X_I(d, t).$$

The minimal pair preserves d, t , changes $proj_I$, and changes export. Therefore k_I is necessary. If no explicit binding records are present, the executable result is:

```
export_binding_count: 0
export_results: []
binding_coordinate_pairwise: []
```

not absence of an application, referent, goal, or world. If I is present, it presents a recorded association among opaque codes. Its tag presents no referent.

The final line is preserved exactly:

*Допущенный вывод не экспортирует себя. Binding должен быть записан отдельно;
записанный binding остаётся записью. Последняя строка также остаётся строкой.*

English gloss:

An accepted derivation does not export itself. A binding must be recorded separately; a recorded binding remains a record. The final line, too, remains a line.

Exact source-form coordinates. The following forms are retained as written objects from the Russian source. Their inclusion records a reading-map coordinate; it neither substitutes them into the English prose nor exports anything the forms name.

Inline mathematical forms:

$$(P, C); (P, C); d; I; I; I; I; ADM_C(P) = 0; X; d, t; P; P, C; d, t; P, C.$$

Inline literal: читает (*reads*).

03. AN AXIOM AND A RULE DO NOT ADMIT THEMSELVES

Parser, checker, compiler, and metaproof coordinates in proof status

Phase 5 • technical study • 29 August 2026

Kirill Shebetov • beforeword / research

A line is presented first.

The rule for admitting it is presented next.

A rule does not discover its admission inside its own line.

Abstract

This file resolves the word доказательство (*proof*) into exact admission coordinates. It distinguishes:

- the byte record;
- tokenization;
- the parser;
- the grammar;
- axiom admission;
- rule admission;
- the proof checker;
- the compiler;
- the runtime;
- the metaproof;
- payload labels;
- application export.

Every status is introduced only by an explicitly written relation. Neither the line аксиома (*axiom*), nor the line правило (*rule*), nor the line валидно (*valid*) performs that relation. The principal result, **ADR-5**—ADMISSION DEPENDENCY READSET—establishes ReadSet necessity: if the same exact record P receives different statuses under two admissible contracts, the distinguishing contract coordinate belongs to the complete ReadSet of the status. Status is not a function of P alone. **NSA-5** locates the start of admission: in a finite positive nonnull admission system, a cycle of axioms, rules, checkers, or metaproofs without an admitted base creates no admitted node in the least closure. An infinite regress does not become a

start of admission either. It merely writes another admission requirement. An inconsistent calculus and explosion are accepted as positive controls. If a contradiction code and the explosion rule are admitted, the checker may accept every formula code in its exact carrier. This specifies the behavior of a calculus under a contract. It does not turn an accepted line into the object it names, introduce external truth, or present a goal.

PREDECESSORS-AS-PREMISES: NONE.

0. Records are presented first

```
axiom
rule
parser
proof
valid
sound
checker accepted
compiler passed
runtime returned 0
meta-proof complete
therefore true
```

Before an exact contract is introduced, these are lines. The line `axiom` does not admit an adjacent line as an axiom. The line `rule` does not authorize a transition. The line `parser` does not parse itself. The line `proof` does not present a proof status. The line `valid` does not perform a check. The line `sound` does not export a result. The line `checker accepted` may occur inside the payload of a rejected object. The line `runtime returned 0` may be an unread comment. The line `meta-proof complete` is not a metaproof witness. The line `therefore true` adds marks. It does not add what those marks name. The zero boundary is:

payload label $\neq$ protected status field.

Even a protected status field requires a producer, a schema, an input binding, and an acceptance contract.

Label convention: every uppercase status or predicate in this file is expository reader gloss except for explicitly named exact checker enums and fences. In particular, `VALID-DERIVATION-IN-C`, `CHECKER-ACCEPTED-UNDER-HASHED-CONTRACT`, `ACCEPTED-PROOF-UNDER-DECLARED-CONTRACT`, `ADMITTED-AXIOM-IN-C`, and the table labels `TOKENIZED/PARSED/AXIOM/RULE/BUILT/RAN/META-ADMITTED` are not values of the verdict schema. Exact runtime `proof_status` accepts only `ACCEPTED`, `REJECTED`, or `UNPARSED`; exact terminal kinds and overall/fence tokens are listed where they are used.

1. Exact carriers

Fix finite, pairwise typed carriers:

B byte strings ,
 T token streams ,
 F formula-codes ,
 D derivation-codes ,
 A axiom-ids ,
 $\mathbb{R}$ rule-ids ,
 M meta-certificate codes ,
 X executable artifact codes ,
 $\mathbb{Z}$ runtime trace codes .

Matching printed lines do not merge sorts. Formula code $r1$ does not become rule id $r1$. Byte string A does not become axiom id A . A runtime trace does not become a derivation code because its payload has an identical hash. A contract has the exact form:

$$C = (\Sigma, \tau, \pi, G, A, R, E_R, Q, \chi, \kappa, \rho, M, \Lambda).$$

Here:

- Σ is the declared byte alphabet and encoding;
- $\tau \subseteq B \times T$ is the tokenizer relation;
- $\pi: T \rightarrow (F \cup D)$ is a deterministic partial parser function;
- G is the grammar code;
- $A \subseteq A \times F$ is the set of admitted axiom instances;
- R is the complete frozen catalog of ground set-premise rule records;
- $E_R \subseteq \{id(r) : r \in R\}$ is the set of enabled rule ids;
- $Q \subseteq F$ is the set of declared target formula codes;
- $\chi \subseteq D \times F$ is the checker-report relation, read only together with ground-step validation;
- $\kappa \subseteq D \times X$ is the compiler/build relation;
- $\rho \subseteq X \times \mathbb{Z}$ is the runtime-trace relation;
- $M \subseteq M \times (A \cup \mathbb{R} \cup \{\chi, \kappa, \rho\})$ contains meta-admission records;
- Λ is the protected status schema.

The components are listed as inputs. Writing them does not certify their application. In the exact Phase 5 runtime, the supplied surface has already been divided into indexed lines: τ is the identity on line records, and π is partial exact line-text/code decoding from the frozen lexicon. The more general tokenizer/parser notation in this file is an explanatory supertype and does not enlarge the executable theorem domain of ADM-5.

2. Protected statuses

For $p \in B$, define separate predicates. Tokenization:

$$TOK_C(p, t) : \Leftrightarrow (p, t) \in \tau.$$

Parsing:

$$PARSE_C(p, d) : \Leftrightarrow \exists t \in T : TOK_C(p, t) \wedge \pi(t) = d.$$

Axiom admission:

$$AX_C(a, f) : \Leftrightarrow (a, f) \in A.$$

Rule admission:

$$RULE_C(r) : \Leftrightarrow r \in R \wedge id(r) \in E_R.$$

Step acceptance:

$$STEP_C(d, i) : \Leftrightarrow \text{node } i \text{ of } d \text{ is grounded by } A \text{ or by } r \in R \text{ with } id(r) \in E_R.$$

Whole-derivation acceptance:

$$VD_C(p, f) : \Leftrightarrow \exists d \in D : PARSE_C(p, d) \wedge (d, f) \in \chi \wedge f = last_C(d) \in Q \wedge \forall i < n_C(d) STEP_C(d, i).$$

Ordered d, i is only a serialization-interface witness here. The exact runtime normalizes each rule definition into a ground record (rule_id, premise_codes, conclusion_code); the order of premises creates no further admission, and there are no substitutions or schema-level metavariables. A generated trace step has a different exact shape: line_index, statement_code, admission_kind, rule_id, premise_witnesses. The ADM-5 certificate equivalence from file 01 applies to that exact ground checker; ADR-5 below is a separate supplementary ReadSet result.

Compiler acceptance:

$$BUILD_C(d, x) : \Leftrightarrow (d, x) \in \kappa.$$

Runtime acceptance:

$$RUN_C(x, z) : \Leftrightarrow (x, z) \in \rho.$$

Meta-admission:

$$META_C(m, y) : \Leftrightarrow (m, y) \in M.$$

No predicate abbreviates to the presence of its corresponding word in p . In particular:

$$VALID \sqsubseteq p \neq VD_C(p, f).$$

And:

$$AXIOM \sqsubseteq p \neq AX_C(a, f).$$

3. ADR-5: ADMISSION DEPENDENCY READSET

Let C be the declared class of contracts. Let the protected public output use the exact runtime enum:

$$V = \{ACCEPTED, REJECTED, UNPARSED\}, V : B \times C \rightarrow V,$$

where:

- **ACCEPTED** means that the ground trace ended in **TARGET-ACCEPTED** under the exact contract;
- **REJECTED** means that a parsed trace failed line admission or the target gate;
- **UNPARSED** means that exact decoding is absent at the first reached terminal index; an earlier admission failure would already have yielded **REJECTED** and stopped traversal.

These three lines are scoped runtime codes. **ACCEPTED** is not renamed **TRUE**, **REJECTED** is not renamed **FALSE**, and **UNPARSED** does not present the absence of an object.

ADR-5

If there exist $p \in B$ and $C_0, C_1 \in C$ such that

$$V(p, C_0) \neq V(p, C_1),$$

then there is no function

$$v : B \rightarrow V$$

satisfying

$$\forall C \in C : v(p) = V(p, C).$$

Consequently,

$$status(p) \neq status(p, C)$$

as interchangeable signatures.

Proof

Suppose such a v exists. Then

$$v(p) = V(p, C_0)$$

and

$$v(p) = V(p, C_1).$$

By transitivity of equality:

$$V(p, C_0) = V(p, C_1).$$

This contradicts the initial difference between the outputs. Thus exact bytes p are insufficient for both results. The distinguishing coordinate must be read. ▢ The theorem does not say that contracts are “unreal.” It says only:

admission status is not contained autonomously in the unchanged line.

4. ReadSet form of ADR-5

Let the contract be decomposed as:

$$C = (c_1, \dots, c_n).$$

For a deterministic procedure F , coordinate c_j belongs to the extensional ReadSet if there exist contracts C, C' , differing only in c_j , and an input p for which

$$F(p, C) \neq F(p, C').$$

Write:

$$c_j \in \text{ReadSet}(F).$$

RS-5

If C, C' differ only in c_j and the output differs, then

$$c_j \in \text{ReadSet}(F).$$

This definition turns the rhetorical phrase контекст важен (*context matters*) into an exact dependency claim. Neither the word контекст (*context*) nor the word зависит (*depends*) carries the proof any longer. An exact witness pair is required.

5. Strict-prefix witness with unchanged lines

Exact payload:

```
a
a -> b
b
```

Both contracts contain the same frozen core:

$$\text{core}(C) = (\text{surface}, \text{lexicon}, \text{grammar}, \text{axiom table}, \text{rule catalog}, \text{target}, \text{checker version}).$$

The sole distinguishing coordinate is:

$$E(C_{MP}) = \{r_{MP}\}, \text{enabled_rule_ids}(C_{NO-MP}) = \emptyset.$$

Contract C_{MP} specifies:

- frozen decoding $a \mapsto A, a \rightarrow b \mapsto IMP_AB, b \mapsto B$;
- axiom table $\{A, IMP_AB\}$;
- full ground rule record $r_{MP} = (\{A, IMP_AB\}, B)$;
- target code B ;
- checker version and schema;
- enabled set $\{r_{MP}\}$, written in the runtime field `enabled_rule_ids`.

Then:

$$V(p, C_{MP}) = \text{ACCEPTED}.$$

Contract C_{NO-MP} preserves the same bytes, surface, lexicon, grammar, axiom table, complete rule catalog, target, and checker version. Only protected `enabled_rule_ids` changes: rule id r_{MP} remains in the catalog but does not belong to the enabled set. Then:

$$V(p, C_{NO-MP}) = \text{REJECTED}.$$

Contract C_{GLYPH} reads $\rightarrow$ as an inert glyph pair. Under this separately declared contract, exact decoding does not construct a derivation input:

$$V(p, C_{GLYPH}) = \text{UNPARSED}.$$

There is no status fork before the final line. The first two lines form the same accepted strict prefix in both principal contracts. Only admission of the final transition to b differs. The bytes have not changed. The word b has not become less like b . One admission coordinate has changed. Thus `enabled_rule_ids`

belongs to the ReadSet. VALID-DERIVATION-IN-C is not thereby renamed TRUE, and REJECTED-BY-C is not renamed FALSE.

6. A parser is not a coordinate-free window

Let the bytes be:

x - x - x

Parser π_0 returns one identifier. Parser π_1 returns a binary subtraction tree. Parser π_2 rejects the line. Parser π_3 preserves it as comment payload. One written surface admits different parse records. The selected parse does not prove an identifier referent. Nor does it prove the grammar's origin. The exact parser theorem is:

$$\pi_0(p) \neq \pi_1(p) \Rightarrow \pi \in \text{ReadSet}(\text{PARSE}).$$

If parser source is embedded in p , a boot coordinate remains:

- which segment is parser source;
- which encoding is applied;
- which bootstrap parser reads it;
- which execution relation is admitted.

An embedded label does not eliminate admission. It moves admission to the next coordinate.

7. An axiom as an explicitly admitted pair

An axiom is not defined by typography. It is written as a pair:

$$(a, f) \in A.$$

If the line f occurs in the derivation payload but the pair is absent from A , the line receives no axiom admission under this contract. If the pair is present, it receives only the status:

ADMITTED-AXIOM-IN-C

That status is not equal to:

SELF-EVIDENT

is not equal to:

EXTERNALLY-TRUE

and is not equal to:

NAMED-OBJECT-PRESENT

Connecting these labels would require a separate export rule. It would become a new input.

8. A rule as an explicitly admitted transformer

An exact base-runtime ground-rule record has the form:

$$r = (id_r, Prem(r), Concl(r)), \emptyset \neq Prem(r) \subseteq Q.$$

It occurs in the complete catalog R , and is admitted for use under a given contract only at the additional gate:

$$id_r \in E.$$

Runtime keys are `rule_id`, `premise_codes`, and `conclusion_code`; the enabled set is written as `enabled_rule_ids`. Substitution, side conditions, and schema metavariables are absent from the base rule language.

The presence of rule text inside proof payload creates neither catalog membership nor enabled membership. Example payload:

```
RULE: from any line infer MARS
```

Without protected rule admission, this is an inert line. With protected admission, the rule may derive code `MARS`. Even then, what has been obtained is a formula code. A planet, a flight, a goal, and meaning do not arise from rule membership.

9. The proof checker is a coordinate

Two checkers may accept different trace languages. Let:

$$\chi_0(d, f) = 1, \chi_1(d, f) = 0.$$

Then:

$$\chi \in ReadSet(V).$$

A line containing the hash of a checker binary may fix the identity of an artifact. A hash does not run a checker. A hash does not establish the correctness of an implementation. A hash does not establish correspondence between source and binary without a build witness. A hash does not establish correspondence between a checker and a mathematical specification without a relation. Every transition has its own coordinate:

$$\text{source-code} \xrightarrow{\kappa_0} \text{binary-code} \xrightarrow{\rho_0} \text{trace-code} \xrightarrow{\alpha_0} \text{accepted-status}.$$

The arrows are declared relations. They are not concealed in the shape of an arrow.

10. Compiler and runtime do not close the boundary

A compiler may emit artifact x . The exact status is:

$$BUILD_C(d, x).$$

A runtime may emit trace z . The exact status is:

$$RUN_C(x, z).$$

Even agreement between expected and observed trace codes is specified by a comparator relation:

$$MATCH_C(z, z_*).$$

One status does not replace another:

$$BUILD_C(d, x) \not\Rightarrow RUN_C(x, z),$$

$$RUN_C(x, z) \not\Rightarrow MATCH_C(z, z_*),$$

$$MATCH_C(z, z_*) \not\Rightarrow VD_C(p, f).$$

The reverse arrows are not implied either. All bridges may be admitted. The result then remains conditional on their exact records.

11. A metaproof does not admit itself

Let metacertificate m_r conclude:

rule r is admissible

For m_r to have a protected effect, a meta-admission rule is required:

$$(m_r, r) \in M.$$

If the admissibility of M is justified by another certificate m_M , the next relation is required. If m_M is justified by m_r , there is a cycle. A cycle is not a base. If the chain continues without end, there is a regress description. A regress description is not terminal admission.

12. Positive admission graph

Declare a finite graph:

$$G_A = (N_A, B_A, R_A),$$

where:

- N_A is the set of admission claim codes;
- B_A is the set of protected admission-base ids;
- R_A is the set of positive nonnull hyperrules.

Every $r \in R_A$ has:

$$Prem(r) \subseteq N_A,$$

$$ReqBase(r) \subseteq B_A,$$

$$Head(r) \in N_A,$$

and

$$|Prem(r)| + |ReqBase(r)| > 0.$$

For $U \subseteq B_A$:

$$T_U(X) = X \cup \{Head(r) : Prem(r) \subseteq X, ReqBase(r) \subseteq U\}.$$

Least closure:

$$Adm(U) = \mu T_U \text{ or } X_0 = \emptyset.$$

The source-form `or` may be read as “from”; the formula remains the exact preserved record.

13. NSA-5: the no-source-free-admission theorem

Let $SCC \subseteq N_A$ be a strongly connected component. Assume:

1. every rule with its head in SCC has at least one claim premise in SCC ;
2. before iteration, no member of SCC belongs to X_0 .

Then:

$$SCC \cap Adm(U) = \emptyset.$$

Proof

At step 0, the intersection is empty by assumption. Suppose it is empty at step k . For a head in SCC to appear at step $k+1$, a rule with its head in SCC must fire. That rule has a premise in SCC . By the induction hypothesis, the premise is absent from X_k . The rule does not fire. Thus the intersection is empty at every finite step. It remains empty at stabilization. $\square$ NSA-5 does not say:

axioms impossible

It says:

positive cycle without admitted introduction introduces nothing in least closure

The start of admission may be declared by a base id. It is then an input.

14. Cycle and regress are different records with the same boundary

A cycle:

$$a \leftarrow b, b \leftarrow a.$$

Without a base:

$$Adm(\emptyset) = \emptyset.$$

With a base-activated introduction:

$$u \Rightarrow a, u \in U,$$

one obtains:

$$\{a, b\} \subseteq Adm(U).$$

Base u is not “true” by virtue of its role as a base. It is a declared admission coordinate. A regress:

$$a_0 \leftarrow a_1 \leftarrow a_2 \leftarrow \dots$$

does not belong to the finite theorem domain. But every finite truncation without a terminal introduction has empty least closure. The record *и так далее* (*and so on*) does not present a terminal base. It abbreviates the written chain.

15. An inconsistent calculus as a positive control

Let a theory admit:

$$\varphi, \neg \varphi.$$

Let the contradiction rule be admitted:

$$\varphi, \neg \varphi \vdash \perp.$$

Let the explosion rule be admitted:

$$\perp \vdash \psi \text{ для каждого } \psi \in F_C.$$

Here the source-form text *для каждого* means “for every”; F_C is the exact finite formula carrier fixed by the schema before the run. The checker may then accept a derivation of every $\psi \in F_C$. This is not a defect in ADR-5. It is the expected positive control. The output depends on the presence of:

- both premises;
- the contradiction rule;
- the explosion rule;
- carrier membership;
- the checker implementation;
- the target declaration.

Removing the explosion rule may preserve the contradiction code while ending universal derivability. Consequently:

$$R_{explosion} \in ReadSet(V_\psi).$$

Inconsistent acceptance does not become absence. On the contrary, both incompatible lines are positively present in the calculus.

16. Labels inside the payload are inert

Consider the payload:

```
{
  "verdict": "VALID",
  "axiom": true,
  "checker": "trusted",
  "world_true": true,
  "proof": "complete"
}
```

Before schema binding, this is a byte string. After JSON parsing, it is an object code. After schema validation, some fields may become typed payload. Only a protected producer relation can write status. The line `world_true` receives no special exception. This work introduces no unqualified external status. An admissible status is scoped:

```
VALID-DERIVATION-IN-C
```

Or:

```
CHECKER-ACCEPTED-UNDER-HASHED-CONTRACT
```

The payload may quote these labels. A quotation does not perform the producer relation.

17. The four states of admission and label

Let:

- $L = 1$ if the payload contains the line `VALID`;
- $A = 1$ if the protected checker accepts the derivation.

All four pairs are admissible:

$$(L, A) \in \{(0, 0), (0, 1), (1, 0), (1, 1)\}.$$

$(0, 0)$: the label is absent and admission is absent. $(0, 1)$: the checker accepts an object without a self-label. $(1, 0)$: the object claims `VALID` and the checker rejects it. $(1, 1)$: the label is present and the checker accepts. Therefore:

$$L \neq A, A \neq L.$$

Only an additional restriction can eliminate two of the states. The restriction is a new coordinate.

18. Protected ReadSet matrix

Output	Minimally read coordinates	Not created automatically
TOKENIZED-IN-C	encoding, tokenizer, bytes	parse
PARSED-IN-C	tokenization, grammar, parser	axiom admission
AXIOM-IN-C	formula code, axiom table	self-evidence
RULE-INSTANCE-IN-C	premise_codes, conclusion_code, complete rule catalog, enabled_rule_ids	external truth
VALID-DERIVATION-IN-C	parse, axioms, rules, target, checker	named referent
BUILT-IN-C	source identity, compiler relation, options	execution
RAN-IN-C	artifact identity, runtime, inputs, trace policy	target match
META-ADMITTED-IN-C	certificate parse, metarules, meta- bases	source-free authority

The matrix specifies dependency expectations. It does not assert that the listed inputs are presented outside the record.

19. Attempts to bypass the boundary

19.1. “The rule is obvious”

очевидно (*obvious*) is an additional line. It does not replace membership in R .

19.2. “Everyone must accept the axiom”

все обязаны (*everyone must*) does not enumerate carriers, specify a quantifier domain, or perform admission.

19.3. “The checker has been checked by a checker”

A second checker relation has appeared. It requires its own contract.

19.4. “The metaproof closes the question”

A metaproof may close an exact target under a metacontract. It does not admit the metacontract without a coordinate.

19.5. “A working compiler proves the rules”

Compiler output and rule admissibility have different types. A bridge may be written. It does not follow from matching words.

19.6. “Practical success makes the proof true”

A practical-success code is another status. Its connection to a proof requires an application binding.

20. Exact falsifiers

The work does not declare itself invulnerable by means of the word теорема (*theorem*). A falsifier boundary is written for every result.

20.1. ADR-5 falsifier

Present:

1. exact p, C_0, C_1 ;
2. certified $V(p, C_0) \neq V(p, C_1)$;
3. a function $v(p)$ simultaneously equal to both different outputs.

Such an object would violate equality. Mere disagreement with the word contract is not a falsifier.

20.2. ReadSet-claim falsifier

Present an implementation that does not read the distinguishing coordinate yet, for inputs identical on its entire ReadSet, issues two different deterministic outputs without hidden state. If a clock, random seed, cache, or environment is found, it belongs to the extended input.

20.3. NSA-5 falsifier

Present a finite positive nonnull instance satisfying both hypotheses, the least closure from bottom, and an SCC member that appears at a finite iteration. A nullary introduction, a preseeded member, a negative premise, a nonleast fixed point, or a hidden base violates the admission gates.

20.4. Label-inertness falsifier

Present a contract in which mere substring occurrence is an explicitly declared producer rule. The label then receives an effect through that rule. This is a positive control, not autonomous force of the letters.

21. Failure classes

The following codes are not theorem falsifiers:

- PARSE-CONTRACT-MISSING;
- AXIOM-TABLE-UNFROZEN;
- RULE-ID-IN-PAYLOAD-ONLY;
- CHECKER-HASH-MISMATCH;
- COMPILER-OPTIONS-UNBOUND;
- RUNTIME-ENVIRONMENT-UNBOUND;
- META-CERTIFICATE-UNPARSED;
- SOURCELESS-CYCLE-CLAIMED-AS-BASE;
- NONLEAST-FIXED-POINT;
- EXPLOSION-RULE-ASSUMED;
- EXTERNAL-EXPORT-UNDECLARED.

They either remove the instance from the exact admission path or locate a missing coordinate.

22. Self-application

The line `ADR-5` does not admit `ADR-5`. The line `NSA-5` does not create a least closure. This file is a written record. Its formulas receive roles inside a declared reading contract. For a checker to accept a formalization, the following are required:

- exact source bytes;
- a grammar;
- a parser;
- carrier declarations;
- a rule implementation;
- tests;
- a protected verdict schema.

Even a successful verdict would have the form:

```
ACCEPTED - PROOF - UNDER - DECLARED - CONTRACT
```

It would not present what is named outside the record. Self-application does not destroy the theorem. It forbids the theorem from assigning itself an exception.

23. Limit of the result

Established:

$\exists C_0, C_1: V(p, C_0) \neq V(p, C_1) \Rightarrow \text{contract-coordinate необходима для общего status function.}$

The preserved source-form conclusion reads: “the contract coordinate is necessary for the general status function.”

Established:

positive source-free SCC $\Rightarrow$ no admission in least closure.

Not established:

- that every contract must be rejected;
- that formal derivations are useless;
- that a compiler cannot build an artifact;
- that a runtime cannot issue a trace;
- that axioms are forbidden;
- that rules are forbidden;
- that a named external object is absent;
- that purpose is absent.

The stronger conclusion is more exact than a total negation:

an axiom, a rule, a parser, a checker, a compiler, a runtime, and a metaproof receive admission only through written coordinates; neither a cycle nor a payload label creates admission without a declared base or protected producer. A declared axiom coordinate may serve as a base precisely as an input to the contract, but not as a self-generating property of a line.

24. Final cut

Lines are written first. Next it is written how the lines are divided into tokens. Next it is written how tokens are assembled into formulas. Next it is written which formulas are to be called axioms. Next it is written which transitions are to be called rules. Next it is written which trace is to be called a proof. Next it is written which checker is to be called accepted. Next it is written which metaproof admits the checker. A cycle of these records does not create the first admission record. It merely returns the reading to what has already been written.

A proof is not inside the lines before there is a rule for reading the lines.

The reading rule is not inside itself before there is a rule for its admission.

Admission remains a coordinate of the record.

PREDECESSORS-AS-PREMISES: NONE.

04. A FORMAL DERIVATION DOES NOT PRESENT A ROCKET

Binding, application, goal, Mars, and meaning after 100/200/300 years

Phase 5 • technical application • 29 August 2026

Kirill Shebetov • beforeword / research

A formula may be accepted as a derivation.

This does not present what the formula names.

A working rocket is likewise a record in this sentence.

Abstract

This file makes the following typed cut:

```
VALID-DERIVATION
-> truth-label
-> application binding
-> executable artifact
-> runtime trace
-> rocket
-> flight
-> Mars
-> goal
-> endorsement
-> meaning
```

The arrows in this record do not perform the transitions. Every arrow requires a separate protected coordinate. The reader-facing expository label retains its scope:

```
VALID-DERIVATION-IN-C
```

This is reader gloss, not a runtime enum: the executable checker writes `proof_status: ACCEPTED`, and the positive terminal witness is `kind: TARGET-ACCEPTED`. The gloss is not renamed `TRUE`. Checker rejection is not renamed `FALSE` either. Application binding is modeled as a separate opaque code. Two bindings must preserve one `derivation_core_sha256`. If, with an unchanged derivation core, they produce different application statuses, the binding coordinate belongs to the `ReadSet`. A recorded binding does not present the object it names. It presents another relation record. Rocket, flight, Mars, goal, and meaning enter the formal model only as typed opaque codes. Their labels receive no exception. The maximal positive control admits the entire engineering chain:

- the derivation is accepted;
- the binding is accepted;

- the artifact is built;
- the runtime trace is accepted;
- the trajectory criterion matches;
- a goal code is present;
- an endorsement code is present.

Even the complete set establishes only declared relations among records. The question *для чего?* (*what for?*) is not derived from the absence of a goal code. Nor is it derived from the presence of a goal code. One hundred, two hundred, and three hundred years are not constructors of a missing type.

PREDECESSORS-AS-PREMISES: NONE.

0. The initial stop at words

The lines are presented first:

```
истина
применение
работает
ракета
летит
Марс
цель
зачем
смысл
100 лет
200 лет
300 лет
```

Their English glosses are, in order: *truth*, *application*, *works*, *rocket*, *flies*, *Mars*, *goal*, *why/what for*, *meaning*, *100 years*, *200 years*, *300 years*. The word *истина* does not present truth. The word *применение* does not perform an application. The word *работает* does not run an artifact. The word *ракета* does not present a rocket. The word *летит* does not present a flight. The word *Марс* does not present a planet. The word *цель* does not select a goal. The word *зачем* does not present an answer. The word *смысл* does not present meaning. The numerals 100, 200, and 300 are lines in a record. They acquire a temporal role only after a horizon binding. No later concession cancels this stop.

1. Typed carriers

Fix pairwise distinct finite sorts:

B byte records,

D derivation codes,

F formula codes ,
J binding codes ,
X executable artifact codes ,
Z runtime trace codes ,
O named-object codes ,
L flight/trajectory codes ,
P destination codes ,
G goal codes ,
E endorsement codes ,
H horizon codes ,
S meaning-ascription codes .

Identical payload strings do not merge sorts.

$$MARS \in F \neq MARS \in P.$$
$$ROCKET \in O \neq \text{named object}.$$

The final right-hand side is itself a supplied phrase. Carrier membership does not escape the carrier.

2. Opaque marks an exact boundary

A code is opaque relative to a checker if the checker may:

- inspect its type tag;
- inspect its canonical bytes;
- inspect equality;
- inspect a digest;
- inspect membership in a declared relation;
- carry the code as payload;

but may not:

- extract the named object from the label;
- take a resemblance between names as a binding;
- create a goal from a destination;

- create endorsement from a goal;
- create meaning from the age of a record.

Opaque does not mean не существует (*does not exist*). It forbids an unwritten operation.

3. Proof core and its digest

Proof contract:

$$C_p = (\Sigma, \tau, \pi, A, R, Q, \chi, \Lambda_p).$$

For bytes $p \in B$, the parser emits derivation code $d \in D$. For target $f \in F$, introduce:

$$VD_{C_p}(p, f).$$

Its public reading is:

VALID-DERIVATION-IN-C

Not:

TRUE

Here h_p denotes exactly the runtime field `derivation_core_sha256`, not a digest of the entire abstract tuple C_p . The checker constructs a named canonical object, not a tuple or array:

```
{
  "surface_sha256": "...",
  "decoding": [],
  "axiom_codes": [],
  "rules": [],
  "enabled_rule_ids": [],
  "target_code": "..."
}
```

Let D_p denote this exact named object. Then:

$$h_p = sha256(canon(D_p)).$$

The canonicalizer specifies object-key order; key-appearance order in source JSON is not a coordinate. The order of members in protected arrays remains part of the value. Here `rules` means the complete normalized rule catalog, not merely the enabled subset. The digest reports the code of a canonical runtime object under the declared algorithm; it is not an injective identity proof. A digest does not present the content of a label. A digest does not run a parser. A digest does not prove an application.

4. Exact binding model

The following finite tuple is exact within this file's application extension; it is not declared to be an input schema of the executable checker. Its frozen `export_bindings` input projection contains the fields:

```
binding_id
reading_contract_id
claim_code
application_code
goal_code
source_target_code
payload
```

The rich coordinates below either project into those opaque codes or remain separately written expository records.

A binding record has type:

$$j = (id, h_p, f, o, x, \ell, m, g, e, h, \beta).$$

Here:

- id is the binding id;
- h_p is the unchanged derivation-core digest;
- $f \in F$ is a formula code;
- $o \in O$ is a named-object code;
- $x \in X$ is an artifact code;
- $\ell \in L$ is a trajectory code;
- $m \in P$ is a destination code;
- $g \in G$ is a goal code;
- $e \in E$ is an endorsement code;
- $h \in H$ is a horizon code;
- β is a finite protected mapping table.

Binding acceptance:

$$BIND_{C_A}(j, h_p, f).$$

Application contract:

$$C_A = (J, \beta, \kappa, \rho, \mu, \Gamma, \Gamma_E, \Omega, \Lambda_A).$$

Here:

- $J \subseteq J$ is the set of admitted binding records;

- β is the code-mapping table;
- κ is the build relation;
- ρ is the runtime relation;
- μ is the trace comparator;
- Γ is goal admission;
- Γ_E is endorsement admission;
- Ω is the horizon/meaning policy;
- Λ_A is the status schema.

Every coordinate is an input. The line β does not itself perform a binding.

The executable Phase 5 checker retains a separately declared public projection: `proof_status` accepts only `ACCEPTED`, `REJECTED`, or `UNPARSED`; `binding export status` accepts only `EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE` or `EXPLICIT-BINDING-RECORD-WITHOUT-ACCEPTED-TRACE`. These runtime labels record co-occurrence records. They do not present a bound referent. The other uppercase labels in this file are qualified exposition-level diagnoses, not promises of new runtime enum fields.

5. A recorded binding does not present a bound object

The following record is presented:

$$(f, o) \in \beta.$$

This establishes membership of a pair in table β . It does not establish:

$$f = o.$$

It does not establish:

$$o = \text{названное}.$$

The exact source-form `названное` may be glossed as “what is named.”

It does not establish:

$$\text{названное} \text{ предъявлено}.$$

The exact source-form phrase may be glossed as “what is named is presented.”

Even the field:

```
{ "object_code": "ROCKET-01" }
```

is first a written field. A schema may accept it as an opaque object code. Schema acceptance does not turn the field into an object outside the schema.

6. BCR-5: the binding-coordinate ReadSet theorem

Let $j_0, j_1 \in J$ satisfy:

$$\text{derivation_core_sha256}(j_0) = \text{derivation_core_sha256}(j_1) = h_p.$$

Let every proof coordinate agree. Let the bindings differ:

$$\beta_0 \neq \beta_1.$$

Let application procedure F_A issue:

$$F_A(h_p, \beta_0) \neq F_A(h_p, \beta_1).$$

Then:

$$\beta \in \text{ReadSet}(F_A).$$

And no function

$$g(h_p)$$

is simultaneously equal to both different outputs.

Proof

If g reads only h_p , its input is identical in the two runs. Its deterministic output must agree. But the outputs differ. Thus the proof core is insufficient. The distinguishing binding belongs to the ReadSet. $\square$
BCR-5 does not say which binding is *правильный* (*correct*). It establishes a dependency.

7. Two exact counterbindings

One proof core and one computed runtime value h_p : Binding j_0 :

```
formula_code      = F-THRUST-BOUND
object_code       = ROCKET-01
artifact_code     = CTRL-A
destination_code  = MARS-CODE
goal_code        = SCIENCE-GOAL
derivation_core_sha256 =
sha256:764eabc20bce36cdbb2cfe93e49eb2c471e6c4ee33ee984a11f286cda0ea4b16
```

Binding j_1 :

```
formula_code      = F-THRUST-BOUND
object_code       = TEST-RIG-04
artifact_code     = CTRL-B
```

```

destination_code  = NONE-CODE
goal_code         = DEMO-GOAL
derivation_core_sha256 =
sha256:764eabc20bce36cdbb2cfe93e49eb2c471e6c4ee33ee984a11f286cda0ea4b16

```

The exact proof bytes have not changed. The proof contract has not changed. The target formula code has not changed. `derivation_core_sha256` has not changed. The opaque bindings have changed. Different application outputs therefore do not reside autonomously in the proof.

8. Null binding

Admit a third record:

$$j_{\emptyset} = (h_p, f, NO-BINDING).$$

Then the status:

```

VALID-DERIVATION-IN-C
APPLICATION-BINDING-UNAVAILABLE

```

is not contradictory. A valid derivation does not, by its type alone, require an application binding. A binding may be added later. That is a revision record. It was not concealed inside the earlier formula.

9. NOC-5: no export without a constructor

Declare a typed positive closure. Claim sorts:

$$N = N_D \dot{\cup} N_B \dot{\cup} N_X \dot{\cup} N_Z \dot{\cup} N_L \dot{\cup} N_G \dot{\cup} N_E \dot{\cup} N_S.$$

Rules have exact typed heads. If:

1. the initial set $X_0 \subseteq N_D$;
2. no enabled rule has a head in $N_B \cup N_X \cup N_Z \cup N_L \cup N_G \cup N_E \cup N_S$;
3. closure is constructed by positive least-fixed-point iteration;

then:

$$Cl(X_0) \cap (N_B \cup N_X \cup N_Z \cup N_L \cup N_G \cup N_E \cup N_S) = \emptyset.$$

Proof

At step 0, outside-sort members are absent. At every subsequent step, a new member appears only as the head of an enabled rule. No such heads exist. Induction preserves the empty intersection. $\square$ This is **NOC-5**. NOC-5 is a supplementary source-free-closure companion to the separating ReadSet result NEX-5 in file 02, not a specialization or redefinition of NEX-5. The theorem applies only to the frozen tuple (N, X_0, R, E) , where E is written in the field `enabled_rule_ids`. Catalog constructors form the declared completion boundary; NOC-5 does not establish the absence of a constructor or of what is

named beyond that boundary. The theorem does not forbid a bridge. It requires a constructor to be written. If two bridge records with a fixed core give different outputs, the distinguishing bridge coordinate is necessary by BCR-5/NEX-5; constant or extensionally equivalent bridges do not by themselves establish such a dependency.

10. Complete chain of conditional statuses

The following predicates are admissible:

$$\begin{aligned}
 D &= VD_{C_p}(p, f), \\
 B &= BIND_{C_A}(j, h_p, f), \\
 X &= BUILD_{C_A}(j, x), \\
 Z &= RUN_{C_A}(x, z), \\
 L &= MATCH_{C_A}(z, \ell), \\
 G &= GOAL_{C_A}(j, g), \\
 E &= ENDORSE_{C_A}(j, e), \\
 S &= MEANING_{C_A}(j, h, s).
 \end{aligned}$$

Every predicate may be accepted. But:

$$\begin{aligned}
 D &\nrightarrow B, \\
 B &\nrightarrow X, \\
 X &\nrightarrow Z, \\
 Z &\nrightarrow L, \\
 L &\nrightarrow G, \\
 G &\nrightarrow E, \\
 E &\nrightarrow S.
 \end{aligned}$$

Every arrow becomes admissible only after an exact bridge is supplied.

11. The four derivation/application states

Let:

- $D = 1$ be the local code VALID-DERIVATION-IN-C;
- $A = 1$ be the local code APPLICATION-CHAIN-ACCEPTED-IN-C.

Four pairs are admissible:

$$(D, A) \in \{(0, 0), (1, 0), (0, 1), (1, 1)\}.$$

Pair	Exact reading
$(0, 0)$	the derivation is not accepted; the application chain is not accepted
$(1, 0)$	the derivation is accepted; the binding or application chain is absent/rejected
$(0, 1)$	application records are accepted independently of the given derivation
$(1, 1)$	both local statuses are accepted

The numbers are audit bits. 1 does not mean TRUE. 0 does not mean FALSE. A connection between the bits requires a policy.

12. The four goal/endorsement states

Let:

- $G = 1$ mean that a protected goal code is present;
- $E = 1$ mean that a protected endorsement code is present.

The following are admissible:

$$(G, E) \in \{(0, 0), (1, 0), (0, 1), (1, 1)\}.$$

$(0, 0)$: neither code is accepted. $(1, 0)$: a goal is written, but endorsement is not. $(0, 1)$: endorsement refers to another goal scope or the instance is invalid. $(1, 1)$: both codes are accepted. The presence of a goal code is not endorsement. The absence of an endorsement code is not an objection. The absence of an objection code is not endorsement.

13. Truth (Истина) as an intermediate payload label

Consider:

VALID-DERIVATION-IN-C
therefore TRUE

The second line is not a consequence of the first without a rule. If a contract introduces:

$$VD_{C_p}(p, f) \Rightarrow TRUE - LABEL_C(f),$$

the result is a local label:

TRUE - LABEL - ASSIGNED - IN - C

It remains scoped. This work introduces no external scope-free truth status. Label assignment does not present a named referent. The exact heading object *Истина* is the Russian written form glossed here as *truth*.

14. A rocket as an opaque code

Declare:

$$o_R = \text{ROCKET} - 01 \in O.$$

This is an exact code. Its canonical bytes can be hashed. Its equality can be checked. Its presence in a binding can be checked. But:

$$o_R \in O \not\equiv \text{названное предъявлено}.$$

The exact source-form phrase *названное предъявлено* may be glossed as “what is named is presented.” The line *ракета есть* (*there is a rocket*) does not change the type. Even the phrase:

ракета существует и летит

(*the rocket exists and flies*) is a line in this file. Its grammatical confidence is not a bridge.

15. Flight as a relation code

Flight record:

$$\ell_R = (o_R, z, t_0, t_1, \text{criterion_id}) \in L.$$

Comparator:

$$\mu(z, \ell_R) = 1.$$

Protected status:

TRAJECTORY - CRITERION - MATCH - IN - C

It is not abbreviated to:

FLIGHT - OUTSIDE - RECORD - PRESENTED

No such constructor exists in the model. Adding its name will not add the constructor.

16. Mars as a destination code

Declare:

$$m_R = \text{MARS} - \text{CODE} \in P.$$

Binding:

$$(o_R, m_R) \in \beta.$$

A trajectory criterion may refer to m_R . A goal code may refer to m_R . Story payload may refer to m_R . The references remain records.

destination-code present $\neq$ destination is purpose.

A destination answers only the question written in its own contract. It does not automatically answer *для чего?* (*what for?*).

17. Maximal engineering concession

Admit all of the following at once:

1. $D=1$;
2. the binding schema is valid;
3. the core digest matches;
4. the compiler relation is accepted;
5. artifact identity is accepted;
6. the runtime trace is accepted;
7. the comparator returns a match;
8. the trajectory code is accepted;
9. the destination code equals MARS-CODE;
10. a goal code is present;
11. an endorsement code is present;
12. a horizon code is present.

No status is subtracted. The result is:

FULL - DECLARED - CHAIN - ACCEPTED - IN - C

This result is stronger than *ничего не произошло* (*nothing happened*). But it is more exact than the phrase *математика предъявила ракету* (*mathematics presented a rocket*). It presents an accepted record chain.

18. Positive controls

18.1. Formal control

With every proof gate satisfied, the checker must accept the exact derivation. If it rejects it, that is an implementation/test failure.

18.2. Binding control

With matching h_p , schema-valid j , and admitted β , binding status must be positive.

18.3. Build control

If $(j, x) \in \kappa$, build status is accepted.

18.4. Runtime control

If $(x, z) \in \rho$, runtime status is accepted.

18.5. Match control

If the comparator relation holds, match status is accepted.

18.6. Goal control

If g occurs in the protected goal field and passes the scope gate, the goal code counts as present. Positive controls forbid the rhetorical erasure of the technical layer. They do not permit cross-type export.

19. Counterbindings

A counterbinding changes only β . The proof core remains byte-identical. Admit:

$$\beta_0(f) = CTRL - A,$$

$$\beta_1(f) = CTRL - B.$$

Let:

$$RUN(CTRL - A) = z_A,$$

$$RUN(CTRL - B) = z_B.$$

If $z_A \neq z_B$, the proof derivation did not distinguish the outputs. The binding distinguished them. Even if $z_A = z_B$, equality of trace codes does not delete the binding coordinate. It remains part of the producer path.

20. A goal is not derived from a trajectory

A trajectory record may match a criterion while the goal record is absent. This gives:

TRAJECTORY - CRITERION - MATCH - IN - C
GOAL - CODE - NOT - PRESENT - IN - DECLARED - SCOPE

The second line is not equal to:

NO - GOAL

A goal record may be present. The result is then:

GOAL - CODE - PRESENT - IN - C

It is not equal to:

GOAL - JUSTIFIED

Justification requires a separate relation.

21. The “What For?” question (для чего?)

The question enters as an exact query code:

$$q_{why} \in F.$$

Answer record:

$$a_{why} \in S.$$

Acceptance:

$$ANSWER_{C_A}(q_{why}, a_{why}).$$

The presence of an answer code does not present meaning. It establishes an answer relation in a contract. The absence of an answer code does not present meaninglessness. It establishes only:

NO - ANSWER - CODE - IN - DECLARED - SCOPE

The exact query string *для чего?* is glossed as *what for?*.

22. No negative from silence

Let:

$$g \notin G_C.$$

This does not introduce:

$$NO - GOAL.$$

Let:

$$e \notin E_C.$$

This does not introduce:

NOT - ENDORSED.

Let:

$$s \notin S_C.$$

This does not introduce:

MEANINGLESS.

A negative code requires its own producer. Silence is not a producer. A timeout is not a silence certificate. Search failure is not exhaustive absence.

23. Horizon records 100/200/300

Declare:

$$h_{100}, h_{200}, h_{300} \in H.$$

The labels:

100-YEARS
200-YEARS
300-YEARS

do not enact the passage of time. They receive a horizon role through Ω . Let the sequence of records be:

$$X_0 \subseteq X_1 \subseteq \dots \subseteq X_{300}.$$

Transition:

$$X_{t+1} = T_C(X_t).$$

If T_C has no head constructor in the meaning sort, then:

$$X_t \cap N_S = \emptyset \text{ для каждого } t \leq 300,$$

subject to:

$$X_0 \cap N_S = \emptyset.$$

The source-form *для каждого* means “for every.” This is induction over record steps. It is not a claim about the named years outside the record.

24. HRZ-5: time is not a type constructor

HRZ-5

For a positive typed transition system, under the base hypothesis $X_0 \cap N_S = \emptyset$ and provided that no enabled rule has a head in N_S :

$$\forall t \leq n: Cl_t(X_0) \cap N_S = \emptyset.$$

Proof

The base case is given by the empty intersection. The induction step reads the heads of enabled rules. No head belongs to N_S . Hence no new meaning code appears. ▢ If a rule is added at step 201, a meaning code may appear. Its origin is then the new rule/base record. The number 201 did not create the rule.

25. Repetition does not add a coordinate

Let one accepted technical record be copied n times. In ledger notation:

$$X_n = (r_1, \dots, r_n), r_i = r \text{ для каждого } 1 \leq i \leq n.$$

The source-form *для каждого* means “for every.” If canonical set semantics deduplicates copies:

$$X_n = \{r\}.$$

If the ledger preserves multiplicity:

$$\text{count}(r) = n.$$

Neither equality nor multiplicity introduces a goal/endorsement/meaning bridge. Repetition may serve as a premise of an explicit threshold rule. The threshold rule is then an input.

26. 100/200/300 years and a rocket

Admit records stating:

```
year_100: rocket flight accepted
year_200: rocket flight accepted
year_300: rocket flight accepted
```

These are three records. Their schema may be checked. Their ordering may be checked. Their digests may be checked. One may derive:

```
THREE-HORIZON-RECORDS-ACCEPTED-IN-C
```

It does not follow that:

MEANING-ACCUMULATED

The latter requires a constructor.

27. Destination, goal, and meaning do not form a ladder

The following is often written:

$$\text{Марс} \Rightarrow \text{цель} \Rightarrow \text{смысл}.$$

The exact source-form terms are *Mars*, *goal*, and *meaning*. In Phase 5, these are three different sorts.

The following states are admissible:

- a destination is present and a goal is absent;
- a destination is absent and a goal is present;
- a goal is present and endorsement is absent;
- endorsement is present and a meaning answer is absent;
- a meaning answer is present without a rocket binding;
- the whole chain is present.

No state is eliminated by the words *очевидно зачем* (*its purpose is obvious*). A restriction is possible. It must be written.

28. The compiler/checker layer as a record

Checker artifact:

$$x_x \in X.$$

Compiler record:

$$c_x = (\text{source_digest}, \text{compiler_digest}, \text{options}, x_x).$$

Runtime record:

$$z_x = (x_x, \text{input_digest}, \text{output_digest}, \text{environment_digest}).$$

This is sufficient for the exact ReadSet boundary. There is no need to enumerate hardware layers indefinitely. If another layer distinguishes outputs, it is added as a coordinate witness. A list does not replace the theorem. The theorem requires only the distinguishing coordinate.

29. Falsifiers

29.1. BCR-5 falsifier

Present identical h_p , different β_0, β_1 , different deterministic outputs, and a function that reads only h_p yet reproduces both outputs without hidden input.

29.2. NOC-5 falsifier

Present a positive least-closure instance with no outside-sort initial member and no outside-sort rule head in which an outside-sort member appears at a finite iteration.

29.3. HRZ-5 falsifier

Present an exact transition system satisfying the hypotheses and the first t at which a meaning code appears without a meaning-head rule or an initial meaning code.

29.4. Opaque-code-boundary falsifier

Write an explicit constructor that turns an object code into a new protected status. The status then arises through the constructor. This confirms the need for a binding. The line *но ракета же есть* (*but the rocket is right there*) is not a schema witness. It is a new line about a rocket.

30. Limits

The file does not establish:

- the absence of the named rocket;
- the absence of the named flight;
- the absence of Mars;
- the absence of goals;
- the absence of endorsement;
- the absence of meaning;
- the impossibility of engineering application;
- the falsity of formal mathematics;
- the origin of mathematics;
- the passage of 100/200/300 years.

It establishes the more exact boundary:

none of these outputs is extracted from VALID-DERIVATION-IN-C without a separate typed coordinate.

And also:

the presence of a recorded binding does not present the bound referent; it presents a binding record.

31. Self-application

This file contains the words:

ракета
полёт
Марс
цель
смысл

Their English glosses are *rocket*, *flight*, *Mars*, *goal*, and *meaning*. The file does not present what they name. The BCR-5 formula does not present a binding outside the file. The NOC-5 formula does not perform closure. The HRZ-5 formula does not enact the passage of time. Accepting the proofs requires a declared proof contract. Applying the outputs requires a binding contract. Connecting the text to a named referent requires a separate relation. The file itself receives no exception.

32. Final cut

First, a formula was written. Next, a rule for deriving it was written. Next, a checker wrote a local acceptance. Next, a binding wrote the name `ракета` (*rocket*). Next, a runtime wrote a trace. Next, a comparator wrote `совпало` (*matched*). Next, a goal field wrote `Марс` (*Mars*). Next, a horizon field wrote `300 лет` (*300 years*). Next, a question wrote `зачем?` (*why? / what for?*). Every step added a record. No step escaped the record by means of its name.

VALID-DERIVATION does not present truth.

A binding does not present the bound object.

A flight does not select a goal.

Three hundred years do not create meaning.

PREDECESSORS-AS-PREMISES: NONE.

05. ADVERSARIAL TESTING AND SELF-APPLICATION OF PHASE 5

PROOF DOES NOT PRESENT THE PROVED

0. Purpose and zero concession to the label

This file attempts to break PSR-5, ADM-5, and NEX-5 within their exact scope.

The written form *доказательство* (proof) first enters as a sequence of signs.

It is not accepted as a presentation of what has been proved.

The written form *теорема* (theorem) first enters as a sequence of signs.

It is not accepted as an established result.

The written form *проверено* (verified) first enters as a sequence of signs.

It is not accepted as a completed verification.

The string `proof_status = ACCEPTED` is a distinguishable code-status.

It does not introduce an unconditional status `PROOF`.

It does not introduce an unconditional status `TRUE`.

It does not introduce a referent.

It does not introduce a world fact.

It does not introduce an application.

It does not introduce a purpose of application.

It does not introduce meaning.

The red team's main object is not a word about proof, but the exact dependence of the result on the input read.

1. Exact object of Phase 5

Fix a finite exact contract

$$C = (\Sigma, F, \delta_C, Ax_C, R_C, E_C, g_C, mode_C).$$

Here Σ is a finite alphabet of code units.

F is a finite sort of formula-codes.

$P = (\ell_0, \dots, \ell_{n-1})$ is a finite sequence of supplied exact line-records.

$\delta_C(i, \ell_i)$ is the declared deterministic partial per-line decoder. At a reached index, it either returns $\varphi_i \in F$ or terminates with UNPARSED.

$Ax_C \subseteq F$ is the exact set of axiom-codes.

R_C is the exact finite catalog of positive ground set-premise rules.

$E_C \subseteq \{id(r) : r \in R_C\}$ is the exact set of `enabled_rule_ids`; only rules with ids in E_C participate in admission.

$g_C \in F$ is the exact goal-code.

$mode_C$ fixes the declared reading mode, goal matching, and permitted syntactic conventions.

This is a mathematical projection, not a list of literal runtime key names and not a claim of byte-identity with a JSON object.

Field map to runtime:

- P corresponds to the protected `surface_record = {surface_id, lines}`;
- F corresponds to ids from `statement_codes`;
- δ_C corresponds to the exact `decoding`, where each item repeats `line_index`, `line_text`, and supplies `statement_code`;
- Ax_C corresponds to `axiom_codes`;
- R_C corresponds to the complete `rules` catalog;
- E_C corresponds to `enabled_rule_ids`;
- g_C corresponds to `target_code`;
- $mode_C$ corresponds to the frozen top-level `contract` policies.

The runtime key `payload` is not part of the protected admission projection.

Each reached decoded line is admitted by exactly one of two routes:

$$LineOK_C(P, i)$$

if $\varphi_i = \delta_C(i, \ell_i) \in Ax_C$, or if there exists an exact ground rule $r \in R_C$ with $id(r) \in E_C$, whose premise codes have accepted witnesses only at indices $j < i$ and whose head is φ_i .

Final admission:

$$\text{Accept}_C(P)$$

holds exactly when traversal reaches the end, the decoder is defined on every supplied line, all lines have admissible witnesses, and the final line matches g_C in the declared goal-mode.

Exact failure precedence is also sequential: at each reached index, a missing decode first yields UNPARSED; a mapped but inadmissible line yields REJECTED; any terminal event stops the run, so a later defect does not rewrite an earlier status. The target is checked only after full traversal.

No payload string substitutes for $\Sigma, F, \delta_C, Ax_C, R_C, E_C, g_C, mode_C$.

2. Admission gates

An attack counts as an attack on Phase 5 only after passing the gates.

2.1. FINITE-EXACT-SURFACE

The exact code units of P are presented.

Their order is presented.

Their boundaries are presented.

Normalization is either prohibited or included in the contract.

2.2. FINITE-TYPED-CONTRACT

Σ, F , axioms, the complete rule catalog, `enabled_rule_ids`, arities, goal, and mode are finite and typed.

A formula-code does not become a rule because of its name.

A rule-code does not become an applied rule because of a payload.

2.3. DECLARED-DETERMINISTIC-PARSER

The parser and parser version are presented as a coordinate of the contract.

If parsing permits several trees, the tie-break is part of the mode.

A hidden parser choice does not pass the gate.

2.4. PRIOR-CITATIONS-ONLY

A rule witness for line i may refer only to indices $j < i$.

A cyclic reference to its own conclusion does not pass the gate.

A reference to a future line does not pass the gate.

2.5. EXACT-AXIOMS

Membership in Ax_C is recomputed.

The field `axiom: true` inside a payload does not create membership.

2.6. EXACT-RULES

A rule name does not replace `exact` `premise_codes`, `conclusion_code`, and membership of the rule id in `enabled_rule_ids`.

Every ground rule is recomputed from the exact catalog record.

Presence of a rule in the full catalog does not make it enabled: its id must belong to E_C .

2.7. EXACT-GOAL

The goal-code is presented separately from the proof surface.

The field `goal_reached: true` does not replace final matching.

2.8. DISTINCT-STATUS-CODES

Parser failure, contract failure, rejected derivation, and accepted derivation are not collapsed.

`proof_status = ACCEPTED` is not shortened to `TRUE`.

2.9. FROZEN-READSET

(P, C) is fixed before the result is viewed, including the complete R_C and the enabled subset E_C .

Post hoc addition of axioms, rules, or a goal means a new contract.

2.10. CERTIFICATE-RECOMPUTATION

The parse digest, line witnesses, and goal witness are recomputed.

A claimed certificate is not accepted on the authority of its own name.

2.11. NEX-TYPED-TARGET

For NEX-5, the external target is set in a separate typed status-space.

The words референт (referent), истина (truth), применение (application), цель (purpose), and смысл (meaning) do not replace this target-space.

2.12. NEX-SEPARATING-COMPLETIONS

Two external completions preserve the same (P, C) but yield different target-statuses.

The difference must be presented in a separate bridge/application coordinate.

2.13. NO-PAYLOAD-AUTHORITY

A self-label does not change protected fields.

A payload hash does not turn the payload into an axiom or rule.

2.14. BOUNDED-CLAIM

The output concerns the exact record and exact contract.

It is not silently extended to all records, all parsers, or the world.

Violation of a gate removes the example from the theorem-domain.

It is not a falsifier of the theorem.

3. The three results to be attacked

3.1. PSR-5 — proof-status readset theorem

Let the same exact surface P have different exact values of the canonical ternary status function $V_C(P) \in \{ACCEPTED, REJECTED, UNPARSED\}$:

$$V_{C_0}(P) \neq V_{C_1}(P).$$

Then a selector reading only P cannot return both results for the ordered inputs (P, C_0) and (P, C_1) .

Therefore, the distinguishing contract-coordinate belongs to the full status readset.

PSR-5 does not say that every pair of contracts must distinguish status.

PSR-5 does not say that bytes are unimportant.

PSR-5 says that bytes are insufficient where the same bytes admit a separating pair of contracts.

3.2. ADM-5 — finite admission certificate

Let $W_C(P) = (T, \tau)$ be the generated output of contract evaluation: finite trace T and terminal_witness τ .

Runtime does not receive $W_C(P)$ as input.

Under the Phase 5 gates:

$$Accept_C(P) \Leftrightarrow CertOK_C(P, W_C(P)) \wedge \tau.kind = TARGET - ACCEPTED.$$

The reference certificate combines `derivation_core_sha256`, the generated trace, and the final terminal witness.

The certificate is finite.

The certificate is checkable relative to (P, C) .

The certificate is not an external semantic attestation.

3.3. NEX-5 — no external export without bridge

Let $ADM_C(P)=1$ and `runtime proof_status = ACCEPTED`. External completions K_0, K_1 preserve this one exact accepted (P, C) , but the export target-status function X differs:

$$X(P, C, K_0) \neq X(P, C, K_1).$$

A selector reading only (P, C) cannot return both target-statuses.

Therefore, the separating bridge/application coordinate belongs to the full target-status readset.

Formal admission by itself does not present this coordinate.

4. Red-team status discipline

Four lines are fixed for each attack.

`RECORD-STATUS` — the exact status of the record.

`ESTABLISHED` — what is computed within the declared object.

`NOT-INTRODUCED` — what did not arise from that computation.

`FALSIFIER-TEST` — what result would actually break the theorem.

The strings `status`, `proof_status`, `terminal_witness.kind`, `export_status`, `comparison_status`, and the four fences below reproduce the checker's exact vocabulary.

Uppercase labels for compiler, signature, title, test, history, and other extended objection contracts are not fields of `proof_readset_checker.py`; they are explicitly scoped record-classifications of the corresponding attack.

They do not substitute for runtime vocabulary.

There is no unqualified `PROOF` below.

There is no unqualified `TRUE` below.

There is no unqualified `FALSE` about the world below.

5. Attack A — same bytes, two parser-contracts

Surface:

```
a
a -> b
b
```

Contract C_0 contains total exact decoding of the three indexed lines into codes $S:A$, $S:I_AB$, $S:B$.

Contract C_1 preserves the same surface-lines, but its partial decoding contains no mapping for the last line.

In C_0 , the first two lines are axioms, and modus ponens is a declared rule.

In C_1 , the checker stops at the first unmapped surface index.

The bytes are identical.

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = UNPARSED; terminal_witness.kind = UNMAPPED-SURFACE-LINE.

ESTABLISHED: the exact decoding-coordinate distinguishes the admission outputs.

NOT-INTRODUCED: neither output introduces a referent of a , a referent of b , or a truth-value outside the contract.

FALSIFIER-TEST: without reading the contract, a bytes-only selector must return two different ordered outputs for one exact P .

Such a selector must simultaneously satisfy $z(P)=0$ and $z(P)=1$.

Presenting only one branch does not falsify PSR-5.

6. Attack B — tokenization variant

Surface:

```
p->q
```

Parser C_0 reads p , $->$, q .

Parser C_1 reads one atom $p->q$.

The goal of C_0 is the implication-code.

The goal of C_1 is the atom-code.

Even if both branches are accepted, their parsed objects differ.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = ACCEPTED.

ESTABLISHED: two contracts can assign two different typed parses to one surface.

NOT-INTRODUCED: the shared word `accepted` does not establish identity of the parsed formula-codes.

FALSIFIER-TEST: PSR-5 is not activated here if the exact acceptance bit is the same.

If the claimed output includes parse identity, the difference again requires the parser-coordinate.

Coincidence of one status bit does not remove the contract from the full certificate readset.

7. Attack C — whitespace and normalization

The surface contains an exact sequence with two spaces.

C_0 ignores repeated whitespace.

C_1 treats the second space as a malformed token.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = UNPARSED.

ESTABLISHED: the normalization policy belongs to the readset.

NOT-INTRODUCED: the word `пробел` (space) does not present physical empty space; a code unit is being counted here.

FALSIFIER-TEST: the same surface-only output must coincide with two different contract-relative results.

Invisible pre-normalization is not an exception.

It is a hidden contract.

8. Attack D — same bytes, different axioms

Surface:

a

In C_0 : $a \in Ax_{C_0}$ and $g_{C_0} = a$.

In C_1 : $a \notin Ax_{C_1}$, rules are empty, and $g_{C_1} = a$.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = REJECTED.

ESTABLISHED: axiom membership is a contract-coordinate.

NOT-INTRODUCED: axiom membership does not present what a names and does not establish world truth.

FALSIFIER-TEST: a function of bytes alone must distinguish the contracts without reading the difference in axioms.

The field AXIOM a inside the surface does not help.

It is first just another surface line.

Only the parser and axiom policy give it a role.

9. Attack E — same bytes, different rules

Surface:

```
a
a -> b
b
```

C_0 contains modus ponens.

C_1 does not contain modus ponens.

The axioms and goal are identical.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = REJECTED.

ESTABLISHED: the rule-set distinguishes admission.

NOT-INTRODUCED: application of modus ponens within the code-system does not bind a or b to objects.

FALSIFIER-TEST: a rule-blind selector must produce two different correct outputs on the same readset.

The name modus ponens is not a rule instance.

Exact premise_codes, earlier accepted witnesses, and conclusion_code are required.

10. Attack F — extension with a different side condition

This attack is not a result of the executable Phase 5 base kernel. The base kernel permits only positive ground set-premise rules without substitution or a side-condition language.

If an extension introduces a side condition, it creates a new protected contract coordinate. Two extended contracts may then contain a rule with the same name R .

In C_0 , the side condition passes.

In C_1 , an additional freshness-condition is required.

The surface is identical.

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = REJECTED.

ESTABLISHED-IN-EXTENSION-ONLY: the rule name does not exhaust the exact extended schema.

NOT-INTRODUCED: equality of names does not establish equality of rules.

FALSIFIER-TEST-FOR-EXTENSION: the new side-condition coordinate must prove unnecessary despite separated outputs.

If the extended checker reads the side condition, it reads the extended contract.

If the base checker does not read it, that is not a defect of the base runtime: the side-condition field is outside its declared input language.

11. Attack G — one parse, different goals

The surface and derivation graph are identical.

In C_0 , the goal is the final formula b .

In C_1 , the goal is c .

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = REJECTED; terminal_witness.kind = TARGET-LINE-MISMATCH.

ESTABLISHED: the goal-coordinate belongs to the admission readset.

NOT-INTRODUCED: reaching a goal-code does not introduce a purpose of action or the purpose of an object.

FALSIFIER-TEST: a goal-blind selector must correctly distinguish the outputs.

The word цель (goal) has only a declared record-role here.

It does not present для чего (what for).

12. Attack H — accepted derivation without binding

The contract admits the proof surface.

The supplied contract records no external binding coordinate.

It contains no map from atom-codes to objects.

It contains no field labelled `truth-assignment`.

It contains no application target.

It contains no purpose coordinate.

RECORD-STATUS: `proof_status = ACCEPTED`.

ESTABLISHED: the generated $W_C(P)$ passes `CertOK`, and the terminal kind is `TARGET-ACCEPTED`.

NOT-INTRODUCED: a referent, world truth-status, application, purpose, and meaning.

FALSIFIER-TEST(NEX-5): a unique external target-status must be computable from one (P, C) without any bridge-coordinate, for every separating completion.

Simply repeating the accepted-status does not meet this test.

It remains on the (P, C) side.

13. Attack I — one accepted proof, two bindings

The same (P, C) is fixed.

Completion K_0 binds atom `r` to target-id `u`.

Completion K_1 binds atom `r` to target-id `v`.

$u \neq v$ in the declared target sort.

Formal admission is the same in both branches.

The external target-status differs.

RECORD-STATUS(K_0): `proof_status = ACCEPTED`; `export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE`.

RECORD-STATUS(K_1): `proof_status = ACCEPTED`; `export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE`.

Pairwise status: `BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS`.

ESTABLISHED: the bridge-coordinate separates target outputs.

NOT-INTRODUCED: the surface does not choose between u and v .

FALSIFIER-TEST: a selector of (P, C) alone must return both differing target-statuses.

If the binding is written into a new contract, the input has already changed.

That confirms rather than removes the readset boundary.

14. Attack J — two truth assignments

One formal derivation is accepted.

Typed completion K_0 assigns the target-code the value τ_0 .

Typed completion K_1 assigns the target-code the value τ_1 .

The values differ as code-members of the target-space.

RECORD-STATUS(K_0): proof_status = ACCEPTED; export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE.

RECORD-STATUS(K_1): proof_status = ACCEPTED; export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE.

Pairwise status: BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS.

ESTABLISHED: formal acceptance is preserved under differing external completions.

NOT-INTRODUCED: acceptance does not choose an assignment.

FALSIFIER-TEST: an assignment-blind selector must be extensionally correct on both branches.

The word истина (truth) in the assignment's name does not attest the world.

The attack distinguishes only declared target-codes.

15. Attack K — inconsistent axioms and explosion

The contract contains axiom-codes p and $\neg p$.

The contract contains the explosion rule.

The surface derives an arbitrary goal-code q .

All line witnesses are correct relative to this contract.

RECORD-STATUS: proof_status = ACCEPTED.

ESTABLISHED: q is admitted in the exact inconsistent calculus.

NOT-INTRODUCED: consistency, target binding, world truth of q , purpose, or meaning.

FALSIFIER-TEST(ADM-5): proof_status = ACCEPTED, but the generated $W_C(P)$ fails CertOK or does not end in TARGET-ACCEPTED.

Explosion does not falsify ADM-5.

It supplies a generated trace within the declared rules.

FALSIFIER-TEST(NEX-5): formal acceptance must independently choose the external target-status in every separating completion.

The word всё (everything) in an informal description of explosion does not expand the target-space.

16. Attack L — paraconsistent variant

The same surface and the same axiom-codes are supplied to C_0 and C_1 .

C_0 contains explosion.

C_1 is a paraconsistent rule-set without explosion.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = REJECTED.

ESTABLISHED: the rule-coordinate again separates the outputs.

NOT-INTRODUCED: the word логика (logic) does not create one uniquely mandatory rule-set.

FALSIFIER-TEST: PSR-5 is broken only by a selector that does not read the rule-set difference and nevertheless returns both branches.

An appeal to familiar logic is a choice of contract, not a property of the bytes.

17. Attack M — circular derivation

The surface claims that line 0 follows from line 0.

The payload contains the label valid proof.

The claimed witness violates PRIOR-CITATIONS-ONLY and is not part of the protected runtime input.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; the recomputed trace either contains only earlier premise_witnesses, or the contract receives proof_status = REJECTED.

ESTABLISHED: the record is not in the domain of the accepted branch of ADM-5.

NOT-INTRODUCED: falsity of what is named, truth of its opposite, or an external semantic status.

FALSIFIER-TEST: a claimed circular witness does not falsify the equivalence; what is needed is a mismatch between `proof_status` and the recomputed generated $W_c(P)$.

A self-reference label does not remove index order.

18. Attack N — future-line citation

Line 0 refers to line 2.

Line 2 refers to line 0.

The parser successfully builds the payload graph.

But the graph does not satisfy the admission mode.

RECORD-STATUS: `proof_status = REJECTED`.

ESTABLISHED: parse success is not line admission.

NOT-INTRODUCED: rejection does not introduce a negative formula-code.

FALSIFIER-TEST: the generated trace contains a premise witness with $j \geq i$, but `CertOK` or runtime nevertheless accepts it under the strict-prefix policy.

If another contract permits cyclic proofs, that is another mode.

PSR-5 applies to a separating pair of modes.

19. Attack O — self-label spoof

The input payload contains:

```
status: proof_status = ACCEPTED
theorem: PROVED
world_truth: true
```

The protected parser, axioms, rules, and goal do not take these fields as authority.

If the spoof is added only to the payload: RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS, and `proof_status` is recomputed from protected fields. If the spoof breaks the protected shape: RECORD-STATUS: INVALID-PROOF-READSET-RECORD.

ESTABLISHED: the self-label does not change the recomputed result.

NOT-INTRODUCED: proof status, theorem result, or world truth from the payload.

FALSIFIER-TEST: the spoof must change the recomputed protected output while the protected readset remains unchanged.

If the schema permits a payload, it is preserved only as an uninterpreted field.

If the schema prohibits a payload, the record is rejected.

Both branches preserve the boundary.

20. Attack P — certificate spoof

The certificate lists an incorrect parse digest.

The line witnesses look syntactically plausible.

The goal witness refers to a missing line.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; the claimed certificate bytes are inert, while the exact `trace` and `terminal_witness` are recomputed.

ESTABLISHED: the claimed certificate bytes are outside the protected readset; generated $W_C(P)$ is recomputed.

NOT-INTRODUCED: external proof authority or semantic status.

FALSIFIER-TEST: ADM-5 is falsified only by a divergence between the exact `proof_status` and `CertOK` of generated $W_C(P)$ with the terminal condition.

One claimed certificate does not change the generated output.

The recomputed generated trace closes only admission relative to (P, C) .

21. Attack Q — checker as a proving machine

Objection: the checker returns `accepted`, so the proved has been presented.

The checker receives serialised (P, C) .

The checker implements the parser and admission algorithm.

It returns a new serialised record.

RECORD-STATUS: CHECKER-OUTPUT-MATCHES-DECLARED-ADMISSION-ALGORITHM.

ESTABLISHED: a reproducible input-output relation within the implementation boundary.

NOT-INTRODUCED: the named object, world truth, implementation infallibility, purpose, or meaning.

FALSIFIER-TEST(ADM-5): recomputed `CertOK` of the generated output and the accepted predicate must diverge on an admitted input.

FALSIFIER-TEST(NEX-5): without a bridge, the checker must return the correct differing external target-statuses for separating completions it does not read.

The label `Verify` on a button does not expand the readset.

22. Attack R — compiler objection

Objection: a compiler turns symbols into working machine code.

A compiler reads source relative to a grammar, type rules, target ISA, and build options.

Changing the options can change the output for the same source.

Changing the ISA can make the output invalid.

RECORD-STATUS: COMPILED-UNDER-DECLARED-TOOLCHAIN-CONTRACT.

ESTABLISHED: the output bytes were produced by the given toolchain pipeline.

NOT-INTRODUCED: referents of source names, world truth of comments, the program's purpose, or the meaning of execution.

FALSIFIER-TEST(PSR-5): a source-only selector must reproduce different toolchain-relative outputs without reading the toolchain-coordinate.

FALSIFIER-TEST(NEX-5): compiled bytes must choose an external application target without a deployment bridge.

Compilation is a later operation on a record.

It does not make source into what the source names.

23. Attack S — proof-assistant objection

Objection: a proof assistant kernel checks the theorem, so a contract is no longer needed.

The kernel itself fixes the parser, term language, typing rules, reduction, and trusted primitives.

Kernel version and flags belong to the operational contract.

RECORD-STATUS: KERNEL-ACCEPTED-UNDER-DECLARED-VERSION-AND-ENVIRONMENT.

ESTABLISHED: the term has an admissible type-code in the given environment.

NOT-INTRODUCED: an external referent of the type-name, world truth, physical application, or meaning.

FALSIFIER-TEST: one term must receive separated version-relative outputs without the version/environment being read.

The phrase `small trusted kernel` describes the size of the TCB.

It does not remove the TCB.

24. Attack T — soundness theorem

Objection: the calculus has a proved soundness theorem, so an accepted formula is true.

The soundness proof is another proof surface under its own contract.

Its conclusion connects syntactic derivability to model-relative satisfaction under exact definitions.

RECORD-STATUS: SOUNDNESS-DERIVATION-ACCEPTED-UNDER-META-CONTRACT.

ESTABLISHED: the meta-contract admits the recorded implication between code-predicates.

NOT-INTRODUCED: a unique external model-binding, world truth, or a referent of the original atoms.

FALSIFIER-TEST(NEX-5): without a bridge, the meta-proof must distinguish target-status in separating external completions.

Moving the contract to the meta-level does not remove the coordinate.

It adds a new record and a new contract.

25. Attack U — categorical semantics

Objection: uniqueness of a model up to isomorphism fixes what is named.

The phrase `с точностью до изоморфизма` (up to isomorphism) already restricts the type of distinguishability.

The signature, axioms, morphisms, and notion of equivalence belong to the contract.

RECORD-STATUS: MODEL-CLASS-CONSTRAINED-UNDER-DECLARED-SEMANTIC-CONTRACT.

ESTABLISHED: the declared structures satisfy the recorded conditions in the given semantics.

NOT-INTRODUCED: one external object as the named, source-origin, purpose, or meaning.

FALSIFIER-TEST: the same formal record must choose a bridge to one external target in the presence of separating bridges.

Isomorphism is a recorded relation within a model.

It does not present the object behind the record.

26. Attack V — mathematical effectiveness

Objection: a calculation predicts a measurement, so the lines carry truth.

Comparison requires a measurement protocol, units, tolerance, alignment, and a matching rule.

These coordinates are not automatically contained in the proof surface.

RECORD-STATUS: PREDICTION-MATCH-UNDER-DECLARED-MEASUREMENT-AND-TOLERANCE-CONTRACT.

ESTABLISHED: two record values fall within the declared comparison relation.

NOT-INTRODUCED: identity of the records with what is named, ultimate purpose, or meaning.

FALSIFIER-TEST(NEX-5): without a measurement bridge, the proof surface must choose different match-statuses in separating protocols.

Agreement of recorded numbers is a presented agreement.

The name of what is measured does not arise from numerical equality.

27. Attack W — physical rocket

Objection: the numbers made it possible to build a rocket, and the rocket flies, so mathematics has presented the proved.

At least five records are conflated here:

1. a formal proof or calculation surface;
2. an engineering interpretation;
3. manufacturing instructions;
4. an observation protocol;
5. a purpose attribution.

Each transition has a separate bridge-coordinate.

RECORD-STATUS: proof_status = ACCEPTED; world_export_status = NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER. The application-link and observation-match are separate scoped records outside the checker.

ESTABLISHED: under presented contracts, the records can be connected by a sequence of declared relations.

NOT-INTRODUCED: the purpose *для чего* (what for), the meaning of the flight, identity of a formula with a rocket, or world authority of a number.

FALSIFIER-TEST(NEX-5): one (P, C) must determine purpose-status for two completions with the same rocket trajectory and different declared purposes.

A physical difference after launch does not cancel the bridge record.

It does not turn lines of calculation into a flying object.

The answer чтобы долететь (in order to reach the destination) is a new purpose-description string.

Even an observed flight does not contain the string для чего (what for) within itself.

28. Attack X — same rocket, two purposes

Fix one technical record and one observed trajectory-record.

Completion K_0 contains the declared purpose-code RESEARCH.

Completion K_1 contains the declared purpose-code WEAPON.

The formal proof and trajectory-record are identical.

RECORD-STATUS(K_0): export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE; goal_code contains one opaque id.

RECORD-STATUS(K_1): export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE; goal_code contains another opaque id.

Pairwise status: BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS.

ESTABLISHED: the purpose-coordinate separates the outputs.

NOT-INTRODUCED: the trajectory does not choose a purpose-code.

FALSIFIER-TEST: a purpose-blind selector must return both different statuses on one (P, C) and one trajectory-record.

The word исследование (research) does not present a purpose outside the completion.

Nor does the word оружие (weapon) present it.

The attack uses only typed separation.

29. Attack Y — historical origin of mathematics

Objection: mathematics arose before language, independently of language, or from nonverbal quantity.

These sentences enter as records about origin.

Phase 5 does not use the opposite historical sentence as a premise.

PSR-5 concerns the present dependence of exact status on an exact contract.

ADM-5 concerns the present finite generated trace and its CertOK relation.

NEX-5 concerns the present missing bridge under separating completions.

RECORD-STATUS: historical_origin_status = NO-HISTORICAL-ORIGIN-ATTESTATION.

ESTABLISHED: the historical branch is not among the premises of the three results.

NOT-INTRODUCED: confirmation or refutation of the origin of mathematics.

FALSIFIER-TEST: a historical claim falsifies Phase 5 only if it entails a countermodel to the exact theorem that passes its gates.

The words младенец (infant), животное (animal), количество (quantity), and невербальное (nonverbal) do not present what they name.

A reference to an experiment remains a citation-record.

It does not replace an exact countermodel to PSR-5, ADM-5, or NEX-5.

30. Attack Z — “quantity exists without a record”

The sentence presents the written form количество существует без записи (quantity exists without a record).

Phase 5 does not derive its negation.

Phase 5 also does not accept it as an established referent.

RECORD-STATUS: world_export_status = NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER.

ESTABLISHED: the theorem-domain is restricted to record-status and readset.

NOT-INTRODUCED: the existence or absence of quantity outside a record.

FALSIFIER-TEST: a counterexample to the exact readset theorem is required, not a repetition of an ontology sentence.

Even if an external item is separately presented, the word количество (quantity) will not become that item.

31. Attack AA — human intuition

Objection: a mathematician sees truth without a formal contract.

The string `видит истинность` (sees truth) is first a description-record.

If intuition yields an admission output, the conditions that distinguish its outputs constitute an operational coordinate.

RECORD-STATUS: INTUITION-ATTRIBUTION-NOT-A-FORMAL-CERTIFICATE.

ESTABLISHED: the attribution does not replace generated $W_C(P)$ and its ADM checks.

NOT-INTRODUCED: access to named truth or the subject's authority.

FALSIFIER-TEST: present a mismatch between `proof_status` and generated $W_C(P)$ in the domain of ADM-5, rather than changing the domain to intuition-mode.

If the mode permits an oracle step, the oracle belongs to the contract.

32. Attack AB — universal agreement

Objection: everyone agrees with the proof.

A list of agreements is a new record family.

The quantifier `все` (everyone) requires a declared finite or otherwise specified domain.

RECORD-STATUS: AGREEMENT-RECORD-SEPARATE-FROM-ADMISSION-RECORD.

ESTABLISHED: with an exact roster, an agreement relation can be computed.

NOT-INTRODUCED: referent, world truth, source, or meaning of the proof surface.

FALSIFIER-TEST: consensus must eliminate separating contracts or bridges without adding a readset-coordinate.

Adding votes does not change the bytes of the original proof.

33. Attack AC — successful repetition

The checker is run a million times.

The output is the same every time.

RECORD-STATUS: DETERMINISTIC-REPETITION-UNDER-FIXED-IMPLEMENTATION.

ESTABLISHED: repeatability of the implementation relation in the tested environment.

NOT-INTRODUCED: an external referent, world truth, complete absence of bugs, or meaning.

FALSIFIER-TEST: repetition must allow the selector to distinguish unread contract or bridge coordinates.

The number of repetitions does not expand the input readset.

34. Attack AD — recorded test-result lines

For the recorded run, the test suite emitted the literal PASS.

Test cases and expected outputs are records.

The runner compares actual and expected under a declared equality.

RECORD-STATUS: TEST-SUITE-PASS-UNDER-DECLARED-RUNNER-AND-CASES.

ESTABLISHED: the tested input-output pairs matched the recorded expectations.

NOT-INTRODUCED: truth of what is named, completeness of the test space, absence of all defects, or external authority.

FALSIFIER-TEST: an admitted counterexample to the theorem is found, not merely an untested implementation branch.

A run emitting the literal FAIL may disclose an implementation defect.

Passing tests do not become theorem premises.

35. Attack AE — hash equality

A SHA-256 digest is published.

Recomputation yields the same digest.

RECORD-STATUS: BYTES-MATCH-RECORDED-DIGEST-UNDER-DECLARED-HASH-ALGORITHM.

ESTABLISHED: the byte sequence matches the hash-addressed release record within the chosen algorithm relation.

NOT-INTRODUCED: truth of the text, authorship, origin, completeness, meaning, or referent.

FALSIFIER-TEST: hash equality must by itself supply an external target-status under separating completions.

A hash fixes the distinguishability of bytes.

It does not make the bytes into the proved.

36. Attack AF — digital signature

The signature passes cryptographic verification.

RECORD-STATUS: SIGNATURE-VALID-UNDER-DECLARED-KEY-AND-ALGORITHM.

ESTABLISHED: the signature relation passes verification for the presented bytes and key record.

NOT-INTRODUCED: truth of the payload, identity outside the key-binding contract, purpose, or meaning.

FALSIFIER-TEST: the signature must replace the contract and bridge without introducing its own coordinates.

The key binding is already an additional record.

37. Attack AG — title authority

Public title:

ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

PROOF DOES NOT PRESENT THE PROVED

The title is a presented string.

It is not a premise of PSR-5.

It is not a certificate of ADM-5.

It is not a bridge of NEX-5.

RECORD-STATUS: UNBOUNDED-STANDALONE.

ESTABLISHED: an exact heading occurrence exists in the release bytes.

NOT-INTRODUCED: unconditional truth of the heading, a world law, or the author's authority.

FALSIFIER-TEST: the heading must be replaced by a countermodel to the exact theorem, not by an opposing heading.

The title itself is subject to its own boundary.

It does not present *доказательство* (proof).

It does not present *доказанное* (the proved).

38. Attack AH — theorem-label self-application

The labels PSR-5, ADM-5, and NEX-5 are code-names.

Their presence does not create theorem status.

Theorem status arises only after definitions, premises, and a derivation are read under a contract.

RECORD-STATUS: THEOREM-LABELS-PRESENT-IN-RECORD.

ESTABLISHED: the labels address exact claim blocks.

NOT-INTRODUCED: truth, novelty, priority, or external validity of these claims.

FALSIFIER-TEST: an exact admitted countermodel to the corresponding claim, passing the gates, is required.

Phase 5 does not exempt its own formulas from contract dependence.

This is not a defect of self-application.

It is the stated boundary.

39. Attack AI — checker-name self-application

The filename `proof_readset_checker.py` does not present checker behavior.

Executable bytes and the run environment are separate inputs.

RECORD-STATUS: FILE-NAME-PRESENT before execution.

RECORD-STATUS: CHECKER-RUN-RECORDED only after an exact command and output record.

ESTABLISHED: in a reproduced run, a contract-relative string was computed.

NOT-INTRODUCED: infallibility, proof authority, or world truth.

FALSIFIER-TEST: implementation output must diverge from the reference predicate on an admitted test; this is an implementation falsifier, not automatically a theorem falsifier.

To falsify the theorem, the divergence must form an exact countermodel to the theorem statement.

40. Attack AJ — test-name self-application

The name `test_phase5_self_audit.py` is first a filename record.

The prefix `test_` does not execute a test.

The string `PASS` in source is not a run result.

RECORD-STATUS: TEST-SOURCE-PRESENT before execution.

RECORD-STATUS: TEST-RUN-PASS-OR-FAIL after the exact run.

ESTABLISHED: the runner result concerns the exact environment and cases.

NOT-INTRODUCED: completeness of the tests or semantic truth of the claims.

FALSIFIER-TEST: a specific failing invariant must be distinguished from a theorem counterexample.

41. Attack AK — manifest self-application

The manifest lists files, sizes, and digests.

Its own name does not prove package completeness.

The package-filelist and manifest policy must be presented separately.

RECORD-STATUS: MANIFEST-CONSISTENT-WITH-DECLARED-PACKAGE-POLICY or a structured failure.

ESTABLISHED: the listed bytes match the recorded digests.

NOT-INTRODUCED: absence of unlisted files beyond the declared package boundary, truth of the manuscript, or priority.

FALSIFIER-TEST: the manifest is not a premise of PSR-5, ADM-5, or NEX-5; its defect falsifies the release-integrity claim to the exact stated extent.

42. Attack AL — self-hash

A file cannot naively contain an ordinary digest of its own final bytes without a separate fixed-point scheme.

Phase 5 does not require such an object.

The manifest excludes its own string or uses a declared packaging rule.

RECORD-STATUS: HASH-SCOPE-DECLARED.

ESTABLISHED: the digest concerns the exact listed byte scope.

NOT-INTRODUCED: self-certifying truth.

FALSIFIER-TEST: a scope mismatch falsifies the hash claim, but not the readset theorems.

43. Attack AM — self-audit claim registry

The self-audit reads exact occurrences of claims.

It checks hashes, required strings, forbidden strings, and declared dependencies.

RECORD-STATUS: SELF-AUDIT-PASS-UNDER-DECLARED-REGISTRY.

ESTABLISHED: registered syntactic invariants are satisfied.

NOT-INTRODUCED: external truth of the registered claims, novelty, priority, or completeness beyond the registry.

FALSIFIER-TEST: a missing occurrence or hash mismatch falsifies the corresponding registry assertion.

The registrar does not become what is registered.

44. Attack AN — author attribution

An author's name string can be printed in document metadata.

RECORD-STATUS: AUTHOR-NAME-OCCURS-IN-METADATA.

ESTABLISHED: the occurrence concerns exact bytes.

NOT-INTRODUCED: physical authorship, sole origin of every thought, or world priority.

FALSIFIER-TEST: an attribution dispute is not a theorem countermodel unless it affects a premise.

Phase 5 does not use attribution as a premise.

45. Attack AO — predecessors and adjacent works

The presence or absence of a citation list does not change the proof-status theorem.

No external author, school, or paper is a premise.

RECORD-STATUS: PREDECESSORS-AS-PREMISES-NONE.

ESTABLISHED: the manuscript's dependency graph requires no external authority node.

NOT-INTRODUCED: absence of adjacent works, novelty, or world priority.

NOVELTY-STATUS: NOT-ASSESSED.

PRIORITY-STATUS: NO-PRIORITY-CLAIM.

FALSIFIER-TEST: finding similar work changes historical positioning, but does not refute the exact theorem without a countermodel.

46. Exact falsification gate for PSR-5

PSR-5 is considered refuted only upon presentation of all the following:

1. One exact byte-identical P .
2. Two valid exact contracts C_0, C_1 .
3. Different literal values of `proof_status` from the exact enum `ACCEPTED` | `REJECTED` | `UNPARSED`; a difference only in `terminal_witness.kind` under the same `proof_status` does not activate PSR-5 divergence.
4. A selector z whose full readset is provably only P .

5. $z(P)$ correctly equals the output of branch C_0 .
6. The same $z(P)$ correctly equals the different output of branch C_1 .
7. The selector does not read the contract directly, indirectly, through a cache, model version, filename, environment, or precompiled table.

Items 5 and 6 require one value to be two different values.

If the difference between contracts leaks into the environment, the readset is not P .

If the outputs coincide, the separating premise is absent.

If the bytes differ, this is not a same-surface pair.

If one contract is invalid, the exact pair does not pass the gate.

The following do not falsify PSR-5:

- one accepted proof;
- one rejected proof;
- one familiar parser;
- consensus about a parser;
- compiler output;
- a physical result;
- historical origin;
- the self-label proof;
- a hash match;
- a test pass.

47. Exact falsification gate for ADM-5

ADM-5 is considered refuted by either of two types of exact counterexample. Both use generated $W_C(P) = (T, \tau)$, not a supplied certificate input.

Type I:

1. (P, C) passes all admission gates.
2. $\text{Accept}_C(P) = 1$ under the exact definition.
3. Generated $W_C(P)$ fails $\text{CertOK}_C(P, W_C(P))$, or $\tau.\text{kind} \neq \text{TARGET} - \text{ACCEPTED}$.

Type II:

1. (P, C) passes all admission gates.
2. Generated $W_C(P)$ passes $CertOK_C(P, W_C(P))$, and $\tau.kind = TARGET - ACCEPTED$.
3. $Accept_C(P) = 0$ under the same exact definition.

The definition may not be changed between items.

A claimed certificate may not be accepted without generation/recomputation.

An infinite derivation may not be used under the finite-derivation gate.

An oracle step absent from the contract may not be introduced.

The goal may not be replaced by an external meaning.

The following do not falsify ADM-5:

- inconsistent axioms;
- explosion;
- a proof without binding;
- a different parser;
- a failing external application;
- an incorrect certificate;
- a checker bug outside the reference predicate;
- the name доказательство (proof).

48. Exact falsification gate for NEX-5

NEX-5 is considered refuted only if the following are presented:

1. One exact (P, C) .
2. Accepted-core gate: $ADM_C(P) = 1$ and $proof_status = ACCEPTED$.
3. A typed external target-space T .
4. Two admissible completions K_0, K_1 .
5. The same accepted (P, C) in both completions.
6. Different target-statuses $\tau_0 \neq \tau_1$.
7. A selector e with a full readset of only (P, C) .
8. $e(P, C) = \tau_0$ in branch K_0 .
9. The same $e(P, C) = \tau_1$ in branch K_1 .

10. Neither bridge, completion, nor environment marker is read by the selector.

If the target-status does not differ, separation is absent.

If K is written into the input, the readset has already been expanded.

If the selector guesses only one branch, universal correctness is absent.

If the external target is untyped, comparison of the outputs is undefined.

The following do not falsify NEX-5:

- an accepted formal proof;
- a soundness proof;
- a working compiler;
- a matching measurement;
- a flying rocket;
- a declared purpose of the rocket;
- consensus;
- a successful test suite;
- a signed release;
- a historical origin claim.

49. Implementation falsifiers

The Phase 5 implementation fails separately from the theorem under any of the following conditions:

1. The parser is nondeterministic under the declared deterministic mode.
2. The parser reads unprotected payload as contract.
3. An accepted line has neither an axiom witness nor an earlier-rule witness.
4. A rule witness refers to a future index.
5. The witness `rule_id` is not in `enabled_rule_ids`.
6. The witness `premise_codes` are not a subset of codes accepted in the strict prefix.
7. The witness `conclusion_code` does not equal the decoded code of the current line.
8. The final goal does not match g_C .
9. `proof_status` = REJECTED, but generated $W_C(P)$ passes CertOK and TARGET-ACCEPTED.
10. `proof_status` = ACCEPTED, but generated $W_C(P)$ fails CertOK or does not receive TARGET-ACCEPTED.
11. `derivation_core_sha256` does not match the recomputed protected projection digest.

12. A claimed status is accepted instead of recomputation.
13. The different-contract fixture does not preserve exact identical surface bytes.
14. The same-contract control unexpectedly distinguishes outputs.
15. An external bridge is read as an axiom.
16. `proof_status = ACCEPTED` is serialised as `TRUE`.
17. An invalid schema results in unstructured acceptance.
18. The hash depends on key order under declared canonical JSON.
19. A repeated run yields a different verdict on the exact same bytes and environment.
20. The audit omits a required contract coordinate.

Every implementation defect must be registered with its own exact status.

An implementation defect is not hidden by the word `warning`.

The theorem does not protect the implementation from error.

The implementation does not become theorem authority.

50. Required positive controls

The red team requires more than rejection cases.

Control P1: axiom-goal accepted.

RECORD-STATUS: `proof_status = ACCEPTED`.

Control P2: modus-ponens derivation accepted.

RECORD-STATUS: `proof_status = ACCEPTED`.

Control P3: same surface under no-rule contract rejected.

RECORD-STATUS: `proof_status = REJECTED`.

Control P4: same surface under alternate parser rejected.

RECORD-STATUS: `proof_status = UNPARSED`.

Control P5: goal mismatch rejected.

RECORD-STATUS: `proof_status = REJECTED`; `terminal_witness.kind = TARGET-LINE-MISMATCH`.

Control P6: malformed record rejected structurally.

RECORD-STATUS: `INVALID-PROOF-READSET-RECORD`.

Control P7: unsupported contract feature rejected structurally.

RECORD-STATUS: INCOMPATIBLE-PROOF-READSET-CONTRACT.

Control P8: generated trace passes reference CertOK.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; proof_status = ACCEPTED; terminal_witness.kind = TARGET-ACCEPTED.

Control P9: supplied/spoofed certificate remains payload-inert.

The claimed certificate is not runtime input; with protected fields unchanged, the result remains VALID-PROOF-READSET-ANALYSIS with recomputed trace and terminal_witness.

Control P10: accepted proof without binding remains formally accepted.

RECORD-STATUS: proof_status = ACCEPTED.

EXTERNAL-STATUS: world_export_status = NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER.

Control P11: two bindings preserve the formal result and split the target result.

RECORD-STATUS: BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS.

Control P12: inconsistent explosion accepted only in the explosion contract.

RECORD-STATUS: proof_status = ACCEPTED.

Control P13: payload spoof does not alter the protected output.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; the protected trace is preserved, although artifact_sha256 may change.

Control P14: exact repeated run is byte-deterministic.

RECORD-STATUS: PROOF-READSET-INVARIANT-AUDIT-PASS after the full audit, including deterministic replay checks.

Control P15: exhaustive invariant audit covers the bounded contract family.

RECORD-STATUS: PROOF-READSET-INVARIANT-AUDIT-PASS.

No positive control receives an external truth fence.

51. Runtime fences

Every accepted output must preserve:

```
semantic_status = NOT-MACHINE-VERIFIED
trace_scope_status = DECLARED-FINITE-TRACE-AND-CONTRACT-ONLY
world_export_status = NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER
historical_origin_status = NO-HISTORICAL-ORIGIN-ATTESTATION
```

The first fence does not deny machine verification of syntax.

It prohibits reading a syntax checker as a semantic oracle.

The second fence restricts the trace to the declared finite surface and contracts.

It does not declare completeness of all proof systems.

The third fence does not declare the absence of binding, application, purpose, or meaning.

It prohibits autonomous export from trace-status.

The fourth fence does not assert historical origin.

It excludes the historical string from runtime attestation.

52. Self-application matrix

Registry class	Exact record-status	What is established	What is not introduced
C-PUBLIC-TITLE	UNBOUNDED-STANDALONE	title bytes are presented	truth of the title
C-THEOREM-ADM-5, C-THEOREM-PSR-5, C-THEOREM-NEX-5	TEXTUAL-THEOREM-IN-DECLARED-METATHEORY	exact theorem texts are presented	external theorem authority
11 C-PR-* claims	VERIFIED-IN-DECLARED-ARTIFACT	bounded artifact observation matched	referent, world truth, completeness
C-NOVELTY	NOT-ASSESSED; priority NO-PRIORITY-CLAIM	absence of novelty/priority attestation in the registry	novelty or priority

For all 16 claims, semantic status equals NOT-MACHINE-VERIFIED.

The matrix applies without exception to the exact frozen registry of Phase 5. None of these 16 claim records receives a status broader than the one specified; the self-audit makes no statement about all release artifacts.

53. Claim registry

Exact frozen ids:

- C-PUBLIC-TITLE;
- C-THEOREM-ADM-5;
- C-THEOREM-PSR-5;
- C-THEOREM-NEX-5;
- C-PR-MP-CONTRACT-DIVERGENCE;

- C-PR-DECODING-DIVERGENCE;
- C-PR-BOTH-VALID;
- C-PR-BOTH-INVALID;
- C-PR-VALID-NO-BINDING;
- C-PR-VALID-TWO-BINDINGS;
- C-PR-ROCKET-TWO-GOALS;
- C-PR-EXPLICIT-EXPLOSION;
- C-PR-PAYLOAD-SPOOF;
- C-PR-TARGET-MISMATCH;
- C-PR-INVALID-DUPLICATE-CONTRACT-ID;
- C-NOVELTY.

Any other id is absent from the frozen self-audit input and receives no status from the Phase 5 self-audit engine.

Forbidden unbounded claims:

математика никогда не работает (mathematics never works).

ни одной истины не существует (not a single truth exists).

никакого количества не существует (no quantity exists).

все доказательства ложны (all proofs are false).

ракета не летит (the rocket does not fly).

применения невозможны (applications are impossible).

смысла не существует (meaning does not exist).

Phase 5 впервые в мире это доказала (Phase 5 was the first in the world to prove this).

The prohibition does not blunt the main edge.

It prevents the exact cut from being replaced by a new unsupported string.

54. Red-team result

All the strongest objections leave the same remainder.

The parser objection reveals the parser-coordinate.

The axiom objection reveals the axiom-coordinate.

The rule objection reveals the rule-coordinate.

The goal objection reveals the goal-coordinate.

The proof-assistant objection reveals the kernel/environment-coordinate.

The compiler objection reveals the toolchain-coordinate.

The soundness objection reveals the meta-contract.

The measurement objection reveals the measurement bridge.

The rocket objection reveals the application, observation, and purpose bridges.

The historical objection does not enter a premise.

The self-label objection remains a payload.

The test objection remains a test record.

The hash objection remains a byte relation.

No attack has presented a selector capable of obtaining two different correct outputs from one and the same full readset.

No attack has presented a mismatch between an accepted finite derivation and `CertOK` of its generated trace with `TARGET-ACCEPTED`.

No attack has presented an external target-status without a bridge under separating completions.

These strings are the result of the declared adversarial set.

They do not declare exhaustion of all possible future attacks.

55. Surface, digest, and registry attacks

55.1. CRLF, LF, and final newline

Surface identity is defined by exact bytes or exact declared normalization.

A file with `CRLF` and a file with `LF` are not called byte-identical.

A file with a final `LF` and a file without a final `LF` are not called byte-identical.

If a normalizer reduces them to one surface, the normalizer and its version belong to the contract.

`RECORD-STATUS: DISTINCT-SURFACE-BYTES` before declared normalization.

`RECORD-STATUS: SAME-NORMALIZED-SURFACE-UNDER-C` after declared normalization.

ESTABLISHED: the identity criterion distinguishes two levels.

NOT-INTRODUCED: semantic equivalence or an external referent.

FALSIFIER-TEST: a same-bytes pair cannot be claimed after hidden newline conversion.

55.2. Empty final line

In the current runtime, every reached supplied line, including `line_text == ""`, must have an exact decoding item; a missing mapping at a reached index yields `UNPARSED`. If an earlier mapped line has already terminated the run as `REJECTED`, the later line is not reached. Runtime has no ignored-empty-line mode.

An ignored token, formula separator, or malformed-line policy is possible only in an explicitly extended parser supertype. Such an extension adds a protected parser-mode coordinate and is not an executable result of the base checker.

RECORD-STATUS: in the base runtime is determined by first-terminal traversal and exact decoding of every reached supplied line; in an extension, by the separately declared parser-mode.

ESTABLISHED: the current empty-line decoding policy belongs to the parsing readset.

NOT-INTRODUCED: emptiness as an object or absence of anything outside the surface.

FALSIFIER-TEST: a parser-blind selector must separate the outputs on one exact surface.

55.3. Unicode homoglyph arrow

The sign `→` and the sequence `->` have different code points.

The sign `->` with a Unicode minus contains yet another sequence.

Visual similarity is not byte equality.

A parser may declare a normalization map.

The map then belongs to the contract.

RECORD-STATUS: `DISTINCT-CODEPOINT-SURFACES` without a map.

ESTABLISHED: glyph appearance does not substitute for codepoint sequence.

NOT-INTRODUCED: a single implication-object behind three renderings.

FALSIFIER-TEST: a same-bytes fixture must compare bytes, not a screenshot.

55.4. Unicode whitespace

ASCII space, non-breaking space, and thin space differ as code points.

A parser policy can merge them.

That merge is an operation of the contract.

RECORD-STATUS: DISTINCT-SURFACE-BYTES before the operation.

ESTABLISHED: the exact surface audit sees the difference.

NOT-INTRODUCED: a difference in meanings.

FALSIFIER-TEST: hidden whitespace canonicalization defeats the byte-identity claim if it is not presented.

55.5. Strict-prefix premise attack

An admissible premise index must satisfy $j < i$.

The condition $j \leq i$ incorrectly permits a self-premise.

The condition j exists incorrectly permits a future premise.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; runtime does not read claimed premise indices and generates only strict-prefix `premise_witnesses`.

ESTABLISHED: the claimed graph does not pass the strict-prefix gate; runtime generates its own witnesses.

NOT-INTRODUCED: impossibility of cyclic proof in every other contract.

FALSIFIER-TEST: a reference checker accepting $j=i$ under the declared strict-prefix mode suffers implementation defeat.

55.6. Goal occurred earlier, but the final line differs

Surface:

```
goal
aux
```

Both lines may be individually admitted.

The goal-code occurs at index 0.

But the mode requires a final-line goal match.

The final line equals `aux`.

RECORD-STATUS: `proof_status = REJECTED`; `terminal_witness.kind = TARGET-LINE-MISMATCH`.

ESTABLISHED: an occurrence of the goal before the end does not replace the final witness.

NOT-INTRODUCED: goal achievement, external purpose, or meaning.

FALSIFIER-TEST: a checker accepting this record in final-line mode falsifies the implementation claim.

55.7. Minimal PSR pair

For the strongest pair-control, the following are fixed identically:

- exact surface bytes;
- alphabet;
- parser and parser version;
- formula sort;
- axioms;
- the complete catalog of rule definitions;
- goal;
- mode;
- payload;
- metadata.

Only the protected coordinate `enabled_rule_ids` differs.

In C_0 , the modus-ponens rule id is enabled.

In C_1 , the same id is disabled.

RECORD-STATUS(C_0): `proof_status` = ACCEPTED.

RECORD-STATUS(C_1): `proof_status` = REJECTED.

ESTABLISHED: one distinguishing coordinate is sufficient for a PSR pair.

NOT-INTRODUCED: universal dependence of every mathematical practice on one specific JSON field.

FALSIFIER-TEST: the selector must not read `enabled_rule_ids` and must nevertheless return both separated outputs.

55.8. Artifact hash and derivation-core digest

In the namespace `proof_readset_checker`, `artifact_sha256` refers to the canonical digest of the parsed artifact-value under the declared canonicalization.

It is not claimed to be a digest of raw file bytes, CRLF presentation, or an arbitrary serialisation of the same JSON value.

In the namespace `phase5_self_audit`, by contrast, the field with the same spelling `artifact_sha256` refers to the exact raw fixture file bytes and freezes the file before the checker call. These are different record contracts. Equality between the two digests is neither required nor claimed.

`surface_sha256` refers to the protected record `{surface_id, lines}`.

`derivation_core_sha256` refers only to the declared protected derivation core: `surface_sha256`, `decoding`, `axiom_codes`, the complete normalized `rules[{rule_id,premise_codes,conclusion_code}]` catalog without per-rule payload, `enabled_rule_ids`, `target_code`.

These digest scopes may not be collapsed.

Changing the payload may change `artifact_sha256` without changing `surface_sha256` OR `derivation_core_sha256`.

Changing a protected rule coordinate changes the canonical core object, after which the checker recomputes `derivation_core_sha256`. Equal SHA-256 values for different core objects would mean a collision; injectivity of the hash model is not asserted. For the frozen minimal pair, different recomputed digests are in fact recorded.

RECORD-STATUS: equality of proof-readset `artifact_sha256`, `surface_sha256`, and `derivation_core_sha256` comprises three different record relations with different scopes; the self-audit raw-file `artifact_sha256` forms a fourth, separately qualified scope.

ESTABLISHED: each digest fixes its declared byte scope.

NOT-INTRODUCED: truth of the derivation, identity of scopes, or authority of the hash.

FALSIFIER-TEST: scope confusion falsifies the integrity implementation claim.

55.9. Duplicate claim object in self-audit input

The claim registry requires the exact declared multiplicity of objects within `self_claims_phase5.json`.

Two claim objects with one id, where one is required, do not pass the audit.

RECORD-STATUS: SELF-AUDIT-DUPLICATE-CLAIM.

ESTABLISHED: the occurrence count diverges from the registry.

NOT-INTRODUCED: falsity of the claim-content.

FALSIFIER-TEST: an audit that emits the literal `PASS` for a duplicate exact claim receives the implementation-defeat classification.

55.10. Extra unregistered claim object

The appearance of a new bounded claim object within `self_claims_phase5.json` requires a frozen registry id. The engine does not scan all text in files 00–05 and does not claim to detect every sentence a reader might call a claim.

RECORD-STATUS: SELF-AUDIT-EXTRA-CLAIM.

ESTABLISHED: the exact registry input contains an extra object/id.

NOT-INTRODUCED: truth or falsehood of the extra claim.

FALSIFIER-TEST: the literal `PASS` in the presence of a forbidden unregistered claim assigns the audit implementation an implementation-defeat classification.

55.11. Nested prohibited string

A prohibited string can be hidden in a nested JSON value, array, comment-like payload, or escaped sequence within the protected `self_claims_phase5.json` tree.

The self-audit must traverse the entire declared self-claim input tree, not only top-level keys. This check is not a free scan of all manuscript sentences.

RECORD-STATUS: SELF-AUDIT-PROHIBITED-STRING-FOUND.

ESTABLISHED: a forbidden occurrence is present within the protected audit scope.

NOT-INTRODUCED: the external referent of this string.

FALSIFIER-TEST: an audit that checks only the top level and emits the literal `PASS` receives an implementation-defeat classification.

55.12. Explicit scope of the pair theorem

PSR-5 quantifies over an exact pair with separating outputs.

It does not quantify in one step over `всей математике` (all mathematics).

It does not assert that all mathematical surfaces have different statuses in all contracts.

It does not assert that no contract can be widely shared.

It establishes the impossibility of a contract-blind selector on the presented separating pair and on every other pair of the same form.

RECORD-STATUS: PAIR-RELATIVE-READSET-THEOREM.

ESTABLISHED: necessity of the distinguishing coordinate under separated outputs.

NOT-INTRODUCED: a historical, sociological, or ontological universal about all mathematics.

FALSIFIER-TEST: present a selector-only counterexample on an exact separating pair; a counter-generalization about *всей математике* (all mathematics) does not address the pair premise.

This boundary does not weaken the argument.

Even one accepted proof does not receive an autonomous status from its own lines if there exists an exact contract-coordinate without which its status is undefined.

56. The final boundary of Phase 5

Доказательство не находится в начертании.

Оно является статусом начертания под контрактом.

И этот статус не предъявляет доказанное вне контракта.

Proof is not found in the inscription.

It is a status of the inscription under a contract.

And that status does not present the proved outside the contract.

Сначала предъявлены линии.

Статус доказательства появляется только после контракта их чтения.

Допуск внутри контракта не предъявляет названное.

The lines are presented first.

Proof status arises only after a contract for reading them.

Admission within the contract does not present what is named.

A working checker establishes checker-output.

A working compiler establishes compiler-output.

A matching measurement establishes a match under the recorded protocol.

A flying rocket establishes a distinction under the recorded observation contract.

An answer to *для чего* (what for) appears only as a new purpose-coordinate.

Neither title, theorem-label, test, nor hash is exempt from this rule.

PREDECESSORS-AS-PREMISES: NONE.

NOVELTY-STATUS: NOT-ASSESSED.

PRIORITY-STATUS: NO-PRIORITY-CLAIM.

A name does not present what is named.

A formula does not present truth.

An admitted conclusion does not present the proved.