
ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

Теорема зависимого статуса доказательства и граница формального
вывода

Кирилл Шебетов · beforeword / research

Phase 5 · technical version 5.0.0 · 29 августа 2026

Содержание

Аннотация

0. До слова «доказательство»
1. Точный объект опровержения
2. Связь Phase 0–5
3. Типизированная модель
4. Exact read-contract
5. Decoding, strict prefix и target
6. Formal status и три digest
7. ADM-5: теорема finite generated trace
8. Минимальная разделяющая пара
9. PSR-5: теорема зависимого proof-status
10. Жёсткая граница квантора PSR-5
11. «Правила тоже доказаны» и meta-contract
12. Несогласованность и explosion
13. NEX-5: граница formal export
14. Accepted не равно external TRUE; rejected не равно external FALSE
15. Ракета: полный положительный control
16. «Для чего?» и 100/200/300 лет
17. Hard scope fences
18. Точные фальсификаторы
19. Runtime status map
20. Формальная самограница
21. Claim/status и dependency
22. Граница притязаний
23. Итоговый нож Phase 5

Статус источников

01. ТЕОРЕМА READSET СТАТУСА ДОКАЗАТЕЛЬСТВА

ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

0. Область результата
1. Exact surface
2. Finite typed reading contract
3. Sequential validator
4. Extensional ReadSet
5. PSR-5
6. Минимальная collision pair
7. ADM-5
8. Дополнительные exact counterexamples
9. Meta-proof

- 10. Gates
- 11. Certificate
- 12. Запрещённые экспорты
- 13. Extensions
- 14. Limits
- 15. Falsifiers
- 16. Самограница
- 17. Итоговый нож Phase 5A

02. ТЕОРЕМА ОТСУТСТВИЯ АВТОНОМНОГО ЭКСПОРТА

ДОПУЩЕННЫЙ ВЫВОД НЕ ПРЕДЪЯВЛЯЕТ ПРИМЕНЕНИЕ

- 0. Область результата
- 1. Fixed accepted derivation core
- 2. Opaque export sorts
- 3. Export binding
- 4. Abstract export function
- 5. Extensional export ReadSet
- 6. NEX-5
- 7. Минимальная binding pair
- 8. Unbound не является отрицательным мировым status
- 9. Записанный binding не предъявляет referent
- 10. Exact counterexamples
- 11. Export gates
- 12. Executable specialization
- 13. Relation к работающему устройству
- 14. Extensions
- 15. Limits
- 16. Falsifiers
- 17. Самограница
- 18. Итоговый нож Phase 5B

03. АКСИОМА И ПРАВИЛО НЕ ДОПУСКАЮТ СЕБЯ

Координаты parser, checker, compiler и meta-proof в статусе доказательства

Аннотация

- 0. Сначала предъявлены записи
- 1. Exact носители
- 2. Защищённые статусы
- 3. ADR-5: ADMISSION DEPENDENCY READSET
- 4. Readset-форма ADR-5
- 5. Strict-prefix witness при неизменных линиях
- 6. Parser не является окном без координат
- 7. Аксиома как явно допущенная пара

8. Правило как явно допущенный transformer
9. Proof-checker является координатой
10. Compiler и runtime не закрывают границу
11. Meta-proof не допускает себя
12. Positive admission graph
13. NSA-5: теорема отсутствия source-free admission
14. Cycle и regress — разные records, одинаковая граница
15. Inconsistent calculus как положительный control
16. Labels внутри payload инертны
17. Четыре состояния admission и label
18. Protected readset matrix
19. Попытки обхода
20. Exact falsifiers
21. Failure classes
22. Самоприменение
23. Предел результата
24. Итоговый нож

04. ФОРМАЛЬНЫЙ ВЫВОД НЕ ПРЕДЪЯВЛЯЕТ РАКЕТУ

Binding, применение, цель, Марс и смысл через 100/200/300 лет

Аннотация

0. Нулевая остановка на словах
1. Typed carriers
2. Оpaque означает точную границу
3. Proof core и ero digest
4. Exact binding model
5. Recorded binding не предъявляет bound object
6. BCR-5: теорема binding-coordinate readset
7. Два exact counterbindings
8. Null binding
9. NOC-5: no export without constructor
10. Полная цепь условных статусов
11. Четыре состояния derivation/application
12. Четыре состояния goal/endorsement
13. «Истина» как промежуточный payload label
14. Ракета как opaque code
15. Полёт как relation code
16. Марс как destination code
17. Максимальная инженерная уступка
18. Positive controls
19. Counterbindings

20. Цель не выводится из траектории
21. Вопрос для чего?
22. No negative from silence
23. Horizon records 100/200/300
24. HRZ-5: время не является type constructor
25. Повторение не добавляет координату
26. 100/200/300 лет и ракета
27. Destination, goal и meaning не образуют ladder
28. Compiler/checker layer как record
29. Falsifiers
30. Limits
31. Самоприменение
32. Итоговый нож

05. ВРАЖДЕБНАЯ ПРОВЕРКА И САМОПРИМЕНЕНИЕ PHASE 5

ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

0. Назначение и нулевая уступка метке
1. Exact object Phase 5
2. Admission gates
3. Три результата, которые требуется атаковать
4. Status discipline красной команды
5. Атака A — одни bytes, два parser-contracts
6. Атака B — tokenization variant
7. Атака C — whitespace и normalization
8. Атака D — одни bytes, разные axioms
9. Атака E — одни bytes, разные rules
10. Атака F — extension с другим side condition
11. Атака G — один parse, разные goals
12. Атака H — accepted derivation без binding
13. Атака I — один accepted proof, два bindings
14. Атака J — два truth assignments
15. Атака K — inconsistent axioms и explosion
16. Атака L — paraconsistent variant
17. Атака M — circular derivation
18. Атака N — future-line citation
19. Атака O — self-label spoof
20. Атака P — certificate spoof
21. Атака Q — checker как доказательная машина
22. Атака R — compiler objection
23. Атака S — proof assistant objection
24. Атака T — soundness theorem

- 25. Атака U — categorical semantics
- 26. Атака V — mathematical effectiveness
- 27. Атака W — физическая ракета
- 28. Атака X — одинаковая ракета, две цели
- 29. Атака Y — historical origin of mathematics
- 30. Атака Z — «количество существует без записи»
- 31. Атака AA — human intuition
- 32. Атака AB — universal agreement
- 33. Атака AC — successful repetition
- 34. Атака AD — tests passed
- 35. Атака AE — hash equality
- 36. Атака AF — digital signature
- 37. Атака AG — title authority
- 38. Атака AH — theorem-label self-application
- 39. Атака AI — checker-name self-application
- 40. Атака AJ — test-name self-application
- 41. Атака AK — manifest self-application
- 42. Атака AL — self-hash
- 43. Атака AM — self-audit claim registry
- 44. Атака AN — author attribution
- 45. Атака AO — predecessors and adjacent works
- 46. Exact falsification gate PSR-5
- 47. Exact falsification gate ADM-5
- 48. Exact falsification gate NEX-5
- 49. Implementation falsifiers
- 50. Обязательные positive controls
- 51. Runtime fences
- 52. Self-application matrix
- 53. Claim registry
- 54. Результат красной команды
- 55. Поверхностные, digest- и registry-атаки
- 56. Финальный нож Phase 5

Аннотация

Фиксируется конечный protected surface-record

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

и exact reading contract

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

под общей executable policy ω . Lines уже предъявлены структурой input; runtime не сегментирует raw bytes. Contract задаёт partial exact decoding line-index/text $\rightarrow$ opaque statement-code, admitted axioms, полную таблицу positive ground rules, enabled_rule_ids и target-code. Code принимается как axiom либо как conclusion включённого rule, чьи premise-codes содержатся среди ранее принятых lines. Последний принятый decoded line обязан совпасть с target.

ADM-5 устанавливает положительный результат: proof_status: ACCEPTED эквивалентен конечному generated trace с TARGET-ACCEPTED и exact line-level witnesses. Trace является output, не дополнительным input. **PSR-5** доказывает: если одна surface получает разные statuses при contracts, различающихся в protected projection только enabled_rule_ids, surface-only selector не возвращает оба статуса. Proof-status не содержится в lines автономно; различающий field входит в readset. **NEX-5** закрывает второй экспорт. Если две typed completions имеют один и тот же accepted (P, C) , где $ADM_C(P) = 1$ и proof_status = ACCEPTED, но разные referent-, truth-assessment-, application-, goal- или meaning-status, selector, читающий только (P, C) , не может корректно вернуть оба результата. Accepted derivation остаётся relation внутри exact contract. Она не предъявляет названное за его пределами. Историческая строка «математика возникла из языка» не используется как premise. Phase 5 устанавливает более жёсткую текущую зависимость: сейчас proof-candidate не получает status без read-contract, а contract-relative status не создаёт внешний target. **Ключевые слова:** граница записи; proof-candidate; read-contract; enabled rule; strict prefix; generated trace; proof-status; meta-contract; explosion; binding; application; цель; смысл; самоприменение.

0. До слова «доказательство»

Сначала предъявлена поверхность:

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

До reading contract не установлено, что:

- предъявленные `line_text` являются logical formulas;
- `a` и `b` являются formula-codes;
- `->` является оператором;
- вторая строка является импликацией;
- третья строка следует из первых двух;
- последняя строка является target;
- поверхность является доказательством;
- любой code именует что-либо вне записи.

Слова формула, правило, следует, истина, математика и доказательство также сначала входят как написанные формы. Их появление не выполняет названную операцию.

Executable object уже содержит `surface_record.lines`. Это структурная граница input, а не результат parsing raw bytes. Checker не утверждает, что line является proposition; exact decoding contract только сопоставляет line-index, exact `line_text` и opaque statement-code.

Нулевая граница:

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

Строка это доказательство добавляет surface line. Она не добавляет decoding, rule table, enabled ids, target, binding, application или цель.

1. Точный объект опровержения

Объект Phase 5 — автономное доказательное притязание:

конечная последовательность знаков якобы содержит proof-status сама по себе и вследствие этого без отдельного перехода предъявляет referent, external truth-assessment, named state, application, purpose или meaning.

Притязание содержит два скрытых экспорта:

1. surface $\rightarrow$ proof-status без exact read-contract;
2. accepted derivation $\rightarrow$ external target без typed bridge.

PSR-5 закрывает первый. NEX-5 закрывает второй. ADM-5 сохраняет внутренний положительный результат:

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

где runtime fields равны proof_status, trace, terminal_witness.

при полном выполнении declared checks. Generated trace не является дополнительной premise.

Phase 5 не утверждает «вычислять нельзя». Она устанавливает:

$$(P, C) \mapsto FormalStatus$$

может быть exact, но

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

без дополнительных coordinates не определён. Полностью опровергается именно автономный перенос. Более широкая строка «вся математика ложна» не имеет exact target и сама требует read-contract.

2. Связь Phase 0–5

Phase 0 потребовала bridge для formal/application/goal. Phase 1 вывела различающий channel в input. Phase 2 локализовала origin среди предков. Phase 3 связала positive membership с finite support tree и отделила цикл от introduction. Phase 4 установила family-relative absence и включила boundary в readset.

Phase 5 применяет тот же контроль к метке доказательство:

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

Все стрелки до ACCEPTED входят в C. Последняя стрелка не появляется из предыдущих.

Совместимость с Phase 4 точна. CBE-4 может иметь accepted derivation под declared meta-contract.

Phase 5 не отменяет family-relative theorem. Она запрещает только новое переименование:

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

Внутренний result сохраняется; внешний незаписанный экспорт отсекается.

3. Типизированная модель

Фиксируются различные types:

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

Совпадение написания элементов не склеивает types. Оpaque code s:v не становится referent. Target-code s:MARS не становится чьей-либо целью.

Surface:

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

Каждая $\ell_i=(i, line_text_i)$ уже является member protected input. Index обозначает позицию, а не время, причинность или authority. Newline segmentation и Unicode normalization внутри runtime не выполняются; exact contract требует unicode_normalization: none.

Пусть $Q \subseteq StatementCode$ — finite exact universe opaque codes. При total decoding:

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

Payload является inert metadata. Написание TRUE, WORLD или PROOF в payload не меняет protected status.

4. Exact read-contract

Reading contract:

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

где:

- id_C — diagnostic reading-contract id;
- δ — partial exact decoding из indexed lines в Q ;
- $A \subseteq Q$ — axiom_codes;
- $G: R \rightarrow (P(Q) \setminus \{\emptyset\}) \times Q$ — full ground-rule definitions;
- $E \subseteq R$ — enabled_rule_ids;
- $t \in Q$ — target_code;
- m_C — inert payload.

Если $G(r)=(S_r, h_r)$, premise S_r является **set** opaque codes. Order и multiplicity premises не несут роли в executable specialization. Rule-id действует только при $r \in E$. Presence definition в G не равно enablement в E .

Общая policy ω фиксирует partial line-index decoding, deterministic left-to-right validation, positive ground-set premises, axiom-first/lexicographic-rule witness и final-target rule.

Runtime well-formedness/relational validation поверх structural Draft 2020-12 schema требует $A \subseteq Q$, $E \subseteq R$, $t \in Q$, $S_r \subseteq Q$, $h_r \in Q$, sorted unique protected ids и exact supported ω . Structural schema alone не проверяет все sorting, cross-reference и supported-policy gates. Nullary introductions вынесены в A .

Слово `contract` не выполняет эти gates. Schema validation — отдельная процедура над exact serialization.

5. Decoding, strict prefix и target

Contract decoding содержит records $\{\text{line_index}, \text{line_text}, \text{statement_code}\}$. Checker обходит lines слева направо. На каждом reached index сначала проверяется exact mapping. Если он отсутствует, checker останавливается:

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

Если mapping есть, checker немедленно проверяет admission этой line. Earlier LINE-NOT-ADMITTED завершает run, поэтому later unmapped index уже не достигается и не меняет status на UNPARSED.

На fully traversed/total-decoding path $D = (q_0, \dots, q_{n-1})$ strict prefix:

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

Точное admission condition:

$$\text{Good}_i(C, P) \Leftrightarrow q_i \in A_C \vee \exists r \in E_C : G_C(r) = (S_r, q_i) \wedge S_r \subseteq H_i.$$

Если Good не выполнен, checker останавливается:

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

Если eligible rules несколько, runtime выбирает lexicographically least rule-id. Для каждого premise-code trace указывает earliest accepted line occurrence. Это deterministic witness policy, а не premise order.

Target acceptance:

$$\text{TargetOK}_C(D) \Leftrightarrow q_{n-1} = t_C.$$

Именно последний **decoded code**, а не его display payload, сравнивается с target.

6. Formal status и три digest

Основной predicate:

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

Exact runtime field:

proof_status: ACCEPTED | REJECTED | UNPARSED

REJECTED означает line-admission failure либо final target mismatch. UNPARSED означает unmapped surface line. Они не являются external negative statuses.

В namespace proof_readset_checker различаются три exact digest:

1. surface_sha256 — digest protected surface-record {surface_id, lines};
2. artifact_sha256 — digest canonical parsed artifact-value, а не произвольных file bytes;
3. derivation_core_sha256 — digest exact canonical projection fields, определяющих derivation status.

Отдельный namespace phase5_self_audit повторно использует field spelling artifact_sha256 только для SHA-256 exact raw fixture file bytes. Это policy freeze файла, а не canonical parsed-value digest из proof-readset verdict. Namespace задаётся окружающим record contract; два поля нельзя подставлять друг вместо друга.

Derivation core exact:

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

Изменение canonical artifact metadata может изменить artifact_sha256, не меняя derivation_core_sha256 и status. Изменение enabled_rule_ids меняет canonical core object, даже если surface_sha256 прежний; checker пересчитывает digest. Равенство SHA-256 для двух разных core objects означало бы hash collision и не выводится из модели. В опубликованной minimal pair два recomputed digest фактически различны. Hash сообщает digest-code canonical object под объявленным algorithm; он не является injective identity proof, proof-status или external truth-status.

7. ADM-5: теорема finite generated trace

Пусть $W_C(P) = (T, \tau)$ — generated output одного contract evaluation: finite trace T и terminal_witness τ .

Runtime не получает W как input и не принимает отдельный certificate file.

Predicate $CertOK_C(P, W)$ в declared metatheory означает: trace совпадает с left-to-right accepted prefix; каждый step имеет аксиом либо enabled-rule witness; premise-witness indices относятся к earlier accepted lines; τ является exact first terminal kind.

ADM-5

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

$$ADM_C(P) \Leftrightarrow CertOK_C(P, W_C(P)) \wedge \tau.kind = TARGET - ACCEPTED.$$

Доказательство внутри declared meta-contract

Если $ADM_C(P)$, total decoded sequence finite, каждый Good выполнен, и last code равен target. Deterministic axiom-first/least-rule policy строит finite trace, а terminal kind равен TARGET-ACCEPTED.

Обратно, CertOK и TARGET-ACCEPTED фиксируют total decoding, admission каждой line и final target equality. Это conjuncts $ADM_C(P)$. $\square$

ADM-5 — positive control. Generated trace может быть проверен как reference certificate, но не является скрытым input.

8. Минимальная разделяющая пара

Protected P_0 одинакова:

```
a
a → b
b
```

Фиксированы surface_record, statement_codes, decoding, axiom_codes, ПОЛНЫЙ rules, target_code, payload и global policy. Diagnostic reading_contract_id различается по schema, но не входит в derivation core.

Exact decoding, где $q_A, q_{A \rightarrow B}, q_B$ обозначают три opaque ids, перечисленные сразу после формулы:

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

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

Знак $\rightarrow$ inert внутри line_text. Вся line mapped в opaque S:A_IMPLIES_B; никакая operator semantics из начертания не извлекается.

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

Full ground-rule definition одинакова:

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

Единственный protected delta:

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

Enabled run:

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

Generated rule step points to earliest occurrences of both premise-codes at indices 0 and 1.

Disabled run:

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

surface_sha256 совпадает. Comparison returns readset_delta_fields: [enabled_rule_ids] and READING-CONTRACT-REQUIRED-FOR-PROOF-STATUS. Только enabled projection различает protected derivation status.

9. PSR-5: теорема зависимого proof-status

Пусть

$$\sigma: \text{SurfaceRecord} \times \text{Contract} \rightarrow \text{FormalStatus}$$

— deterministic status function.

PSR-5

Если

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

то не существует surface-only selector f , который одновременно возвращает оба exact status:

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

Доказательство внутри declared meta-contract

Function f получает одинаковый argument P , поэтому возвращает одно значение. Premise требует два различных значения. Одно значение не равно двум distinct status-codes. $\square$

PSR-READSET-5

Если contracts различаются только coordinate E , записанной runtime field enabled_rule_ids, и status различается, любое корректное procedure должно читать E либо exact equivalent information.

PSR-NO-INTRINSIC-BIT-5

На declared contract class $\{C_{ON}, C_{OFF}\}$ нет contract-free predicate, который извлекает correct proof-status из P_0 одного. Повторное чтение линий не восстанавливает disabled bit. Слово очевидно его не включает.

10. Жёсткая граница квантора PSR-5

Разделяющая пара доказывает readset necessity в **declared model** и **declared contract class**. Она не является квантором по всем объектам, когда-либо названным словом математика. Это не смягчение ножа. Это защита от того же ложного экспорта, который работа отсекает.

Нельзя вывести:

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

Можно вывести:

Для любого contract class, содержащего separating pair с одним P и разными required statuses, proof-status не является function только P на этом классе.

Это theorem schema с точной premise. Если новая family содержит separating pair, результат применяется. Если family не предъявлена, слово вся не создаёт её.

Тем самым title получает bounded, но окончательный target:

доказательство как автономное свойство линий опровергнуто на каждом классе, где unchanged lines получают разные statuses из-за различающего read-contract.

11. «Правила тоже доказаны» и meta-contract

Пусть C_0 содержит rule-id r . Surface Q_1 завершается code $\text{JUSTIFY}(r, C_0)$. Чтобы принять Q_1 , нужен meta-contract C_1 :

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

Это добавляет accepted derivation под C_1 . Оно не удаляет C_1 и не помещает r в исходную surface.

Если rules C_1 требуют следующего justification, вводятся Q_2, C_2 . Finite chain:

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

META-BASE-5

Для finite meta-chain:

1. последний предъявленный C_m принят как stopping admission текущего audit; либо
2. требование C_{m+1} остаётся непредъявленным внутри chain.

У finite nonempty sequence есть последний index. Если следующего requirement нет, chain остановлен на admission. Если requirement есть, а следующий pair отсутствует, requirement open.

Третьего finite case нет. ▫ Formal practice может честно остановиться на axioms, rule table, trusted kernel или execution contract. Опровергается только фраза «цепь доказательств устранила собственный contract». Infinite regress не требуется для PSR-5. Достаточно одной separating pair.

12. Несогласованность и explosion

Пусть Q содержит distinct polar codes q^+, q^- . Их совместное admission само не создаёт conflict policy.

Explosive ground rules могут включать:

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

для каждого declared $z \in Q$. Если $r_z \in E$, code z получает strict-prefix witness. Paraconsistent contract может иметь те же Q, A, G, t , но $r_z \notin E$. Тогда тот же final z rejected при отсутствии другого witness. Это ещё одна PSR separating pair: меняется enabled id, а не lines. Explosion устанавливает достижимость arbitrary **declared code** под exact rules. Он не предъявляет arbitrary named fact.

Отсюда:

accepted after explosion $\neq$ external establishment.

Чем шире formal rule admits outputs, тем очевиднее typed boundary; payload результата не меняет его origin.

13. NEX-5: граница formal export

Фиксируется target sort

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

Typed completion:

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

где β_x — binding/evaluation/application bridge, а x — target-coordinate.

Formal projection:

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

Target procedure:

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

NEX-5

Фиксируется accepted core: $ADM_C(P) = 1$ и runtime proof_status = ACCEPTED. Если admissible completions Z_0, Z_1 этого же accepted (P, C) удовлетворяют

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

и

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

то нет selector e_X , читающего только (P, C) и корректно возвращающего оба target-status:

$$\nexists e_X : e_X(P, C) = \chi_X(Z_0) \wedge e_X(P, C) = \chi_X(Z_1).$$

Доказательство: projection одинакова, поэтому single-valued e_X возвращает одно значение; required target-status различны. □ NEX-5 не запрещает bridge. Он доказывает, что различающий bridge входит в readset.

Executable specialization использует только record

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

Последние три coordinates записаны runtime fields claim_code, application_code, goal_code.

Checker сообщает co-occurrence B с accepted trace и сравнивает binding coordinates. Он не устанавливает referent или validity связи. Поэтому executable BCR-5 является finite witness readset necessity, а более общий NEX-5 остаётся textual theorem под semantic_status: NOT-MACHINE-VERIFIED.

14. Accepted не равно external TRUE; rejected не равно external FALSE

Граница должна быть записана без эвфемизма:

$$VALID - DERIVATION \neq TRUE,$$

$$INVALID - DERIVATION \neq FALSE.$$

Правые tokens здесь предъявлены только как labels, которые обычно незаконно экспортируют status. Runtime Phase 5 не использует их как unqualified external verdicts.

Точная первая строка:

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

Точная вторая:

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

Rejected derivation не делает payload внешне ложным. Accepted derivation не делает payload внешне истинным. Для external truth-assessment нужен β_T .

Даже formula valid во всех members exact model class M получает bounded status:

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

Слово все не расширяет M , а formula-code не предъявляет structure.

15. Ракета: полный положительный control

Допускается chain:

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

Каждый переход может быть executable и reproducible. Phase 5 это не отрицает. Но calculation status имеет type *FormalStatus*. Controller artifact, telemetry и criterion имеют другие types. Связи входят в application bridge.

Если telemetry удовлетворяет exact criterion, допустим status:

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

Это bounded сильный result. Он не помещается задним числом внутрь математических линий. Surface

ракета полетела

сама остаётся surface до declared decoding и application contract. Слово ракета не предъявляет rocket referent. Работая implementation добавляет records и relations. Она не доказывает, что proof-status был автономным свойством начертания.

16. «Для чего?» и 100/200/300 лет

Formal target t_C и purpose-record $g \in GoalRecord$ различны.

Code REACH-MARS может быть target и accepted. Это не устанавливает:

- чья это цель;
- почему она выбрана;
- должна ли она продолжаться;
- какой meaning ей приписан;
- что flight criterion отвечает для чего.

Purpose completion требует как минимум:

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

Строки

```
100 лет
200 лет
300 лет
```

являются finite surfaces. Numerals не создают duration, future state, purpose или meaning. Temporal constructor входит отдельным input. Повторный calculation может повторить formal output. Это не добавляет goal coordinate. Code БЕССМЫСЛЕННО может быть accepted по rule. Это membership opaque code, а не absence meaning. Nonmembership meaning-code также не является negative meaning-status: polarity boundary Phase 4 сохраняется.

Точный knife:

Ракета может удовлетворить flight criterion. Derivation может удовлетворить formal criterion. Ни один status не содержит ответа «для чего», если purpose/meaning coordinate не введена отдельно.

17. Hard scope fences

Admission gates:

- protected $P = (surface_i d, lines)$ с 1–32 exact lines;
- finite Q, A, G, E , не более 32 statement-codes и 64 rules;
- partial exact decoding by line index/text;
- every enabled id defined;
- positive ground **set** premises, без order/multiplicity;
- prior accepted lines only;
- final decoded line equals target;
- deterministic axiom-first/lexicographic-rule witness;

- supported beforeword.proof-readset.v1 policy;
- для NEX-5 — typed target и separating completions.

Вне gates:

- actual infinite derivations;
- raw-byte segmentation и silent Unicode normalization;
- nondeterministic decoding без declared selection policy;
- cyclic proof graphs;
- oracle admission;
- probabilistic status;
- defeasible/revision rules;
- implicit rule schemas.

Такой object требует расширенной модели. Он не является контрпримером unchanged Phase 5.

Runtime fences:

```
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 в fences являются protocol codes. Они не предъявляют absence вне проверенного field.

18. Точные фальсификаторы

- **ADM-SOUNDNESS-FALSIFIER:** generated trace имеет TARGET-ACCEPTED при unmapped line, bad prefix, missing enabled rule или target mismatch.
- **ADM-COMPLETENESS-FALSIFIER:** $ADM_C(P)$ выполнен, но deterministic generated trace не заканчивается TARGET-ACCEPTED.
- **PSR-FUNCTION-FALSIFIER:** single-valued surface-only f равен двум distinct required statuses.
- **PSR-READSET-SPOOF:** ON/OFF runs различены без чтения enabled id либо equivalent information.
- **MINIMAL-PAIR-DRIFT:** кроме enabled_rule_ids изменён protected decoding, axiom_codes, rules или target_code.
- **ARROW-SEMANTIC-SPOOF:** glyph → получил operator role вне declared opaque decoding.
- **CORE-HASH-CONFLATION:** artifact_sha256 выдан за derivation_core_sha256.
- **STRICT-PREFIX-SPOOF:** current/future line использована как earlier witness.
- **DET-FALSIFIER:** identical core/checker/environment возвращают разные deterministic status.

- **META-BASE-FALSIFIER:** finite chain не имеет last presented index.
- **EXPLOSION-SPOOF:** arbitrary code admitted без enabled explosion либо другого witness.
- **NEX-PROJECTION-FALSIFIER:** formal-only selector возвращает два distinct completion statuses для одного (P, C) .
- **REFERENT-EXPORT-SPOOF:** opaque code повышен до referent без binding.
- **APPLICATION-EXPORT-SPOOF:** formal acceptance повышен до application status без bridge.
- **PURPOSE-EXPORT-SPOOF:** target-code повышен до purpose/meaning без coordinate.
- **PAYLOAD-SPOOF:** protected status меняется только из-за написания payload.

Theorem-falsifier действует только при unchanged gates. Gate drift создаёт другой object.

19. Runtime status map

Exact overall status:

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

Exact proof_status:

```
ACCEPTED
REJECTED
UNPARSED
```

Единственный reader mapping:

Exact runtime token	Развёрнутое чтение
ACCEPTED	все lines decoded/admitted; final accepted code равен target
REJECTED	first reached line mapped, но не admitted; либо все lines reached/admitted, но final target gate не выполнен
UNPARSED	first reached line отсутствует в declared decoding; earlier rejection уже остановила бы traversal

Развёрнутые фразы не являются machine fields.

Exact terminal_witness.kind:

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

Exact pairwise comparison:

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

Exact binding-record co-occurrence:

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

Overall status не является proof_status. proof_status не является terminal kind. Co-occurrence status не утверждает validity binding.

Public title без subtitle и definitions получает:

UNBOUNDED-STANDALONE

20. Формальная самограница

Эта рукопись — finite surface P_M . Слова теорема, доказательство и опровергнуто не дают ей исключения.

Под declared manuscript meta-contract можно проверить:

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

Допустимое reader-description: derivation этой рукописи может быть проверена под declared metacontract. Эта фраза не является runtime machine field и не предъявляет external referent, historical origin, priority или authority.

Если manuscript требовала бы автономного external export, она нарушила бы NEX-5. Если она отрицала бы internal admission, она разрушила бы ADM-5. Самограница сохраняет оба:

exact internal admission retained ,

unwritten external export rejected .

Строка «Phase 5 опровергает всю математику» без определения математика получает UNBOUNDED-STANDALONE. Exact result:

Ни одна surface не содержит contract-free proof-status на классе contracts, который различает её status; ни один accepted formal status сам не предъявляет referent, external truth-assessment, application, purpose или meaning при separating completions.

21. Claim/status и 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
```

Это exact excerpt frozen self-audit registry. Выражения autonomous proof-status и autonomous external export являются conceptual exposition результатов PSR-5/NEX-5, а не claim ids, status values или objects внутри self_claims_phase5.json; self-audit не присваивает им отдельный machine status.

Ни один внешний автор, школа, журнал или термин не является 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 не устанавливает наличие или отсутствие предшественников. Он сообщает только: external texts не входят в derivation premises.

22. Граница притязаний

Установлено внутри declared model

1. Formal admission определяется для (P, C) , а не P отдельно.
2. ADM-5 связывает acceptance с finite generated trace и terminal witness.
3. Minimal pair меняет только enabled_rule_ids и меняет status.
4. PSR-5 исключает correct surface-only selector на separating class.
5. Enabled-rule field входит в readset.
6. Arrow в minimal pair inert и mapped в opaque code.
7. Artifact hash и derivation core digest различены.
8. Finite meta-chain останавливается на admission либо open requirement.
9. Explosion выводит declared code только через enabled ground-rule.
10. NEX-5 исключает formal-only export при separating completions.
11. Referent, truth-assessment, application, purpose и meaning требуют bridges.
12. Flight criterion совместим с theorem и не отвечает для чего.

13. Historical origin не требуется.

14. Самоприменение сохраняет internal result и external fence.

Не установлено этим record

- что formal checking невозможно;
- что arithmetic outputs не reproducible;
- что конкретный referent существует или отсутствует;
- какой external truth-assessment выбрать;
- что implementation работает или не работает;
- что named rocket летит или не летит;
- что у действия есть или отсутствует цель;
- что meaning существует или отсутствует;
- что contradiction обязательно explosive;
- correctness infinite, probabilistic, defeasible or oracle systems;
- theorem о каждом object, названном СЛОВОМ математика;
- historical origin математики;
- global novelty или priority Phase 5.

Это не уступка внешнему. Это запрет расширить exact theorem незаписанным квантором.

23. Итоговый нож Phase 5

Finite surface-record может быть полностью decoded. Opaque code sequence может быть admitted. Каждая non-axiom line может иметь prior-line ground-set witness. Final accepted code может точно совпасть с target. Runtime может сгенерировать полный trace.

Всё это сохраняется:

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

Но одна surface получает ACCEPTED при enabled R:MP и REJECTED при disabled R:MP. Decoding, statement codes, axioms, full rules, target и payload не меняются.

Следовательно:

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

И один formal result совместим с различными external completions. Следовательно:

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

Ракета может удовлетворить flight criterion. Calculation может быть reproducible. Строки 100, 200, 300 лет могут войти в новые records. Ни один из этих results не помещает purpose внутрь исходных линий.

Доказательство не находится в начертании.

Оно является статусом начертания под контрактом.

И этот статус не предъявляет доказанное вне контракта.

Статус источников

Работа продолжает внутреннюю авторскую линию beforeword и Phase 0–4. Внешние сравнительные источники в premises не включены. Историческое происхождение, global novelty и мировой приоритет этим record не устанавливаются.

01. ТЕОРЕМА READSET СТАТУСА ДОКАЗАТЕЛЬСТВА

ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

0. Область результата

Этот файл задаёт конечную exhibited family записей Phase 5A. Он не говорит обо всём, что когда-либо называли словом математика, и не вводит историческое утверждение о происхождении математики. Сначала предъявляется exact surface P . Затем предъявляется finite typed reading contract C . Затем последовательный validator возвращает status-code. Строки

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

остаются строками. Их наличие не предъявляет доказательство, теорему, истину или математику и не повышает status окружающей записи. Название ТЕОРЕМА в заголовке также не даёт себе authority. Теоремный статус PSR-5 ограничен declared model, collision pair и записанным выводом.

1. Exact surface P

1.1. Каноническая surface record

Exact surface является структурой

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

Каждая ℓ_i — exact text line: Unicode scalar sequence без silent normalization. Равенство surfaces означает равенство surface_id, числа lines и каждой line по scalar sequence.

Raw JSON whitespace, order object keys и package filename не являются semantic fields P . Canonical serialization задаёт:

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

Digest является code canonical record. Он не предъявляет автора, источник или названное содержанием P .

1.2. Payload

Raw payload можно хранить, показывать и хешировать. Он не изменяет protected contract fields, пока explicit constructor контракта не задаёт такое изменение. Строка

THIS IS A VALID PROOF

не является shortcut к acceptance. Если она входит в P , она проходит тот же decoder и те же admission gates, что остальные строки. Metadata вне P, C имеет noninterference condition:

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

2. Finite typed reading contract C

2.1. Защищённые namespaces

Фиксируются конечные disjoint sorts:

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

Q — claim-codes, R — ground-rule ids, T — token-codes. Tagged code не предъявляет названное своим tag.

2.2. Поля

Contract имеет вид

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

Здесь:

- id_C — diagnostic reading_contract_id;
- L — fixed exact-text lexicon;
- δ — deterministic partial decoder indexed exact line-records: $(i, \ell_i) \mapsto Q$;
- Q — complete code universe этой instance;
- R — complete rule-id universe;
- $A \subseteq Q$ — admitted assumption-codes;
- $D: R \rightarrow (P(Q) \setminus \{\emptyset\}) \times Q$ — full ground-rule definitions с непустым множеством premises;
- $E \subseteq R$ — enabled_rule_ids;
- $t \in Q$ — target-code;
- ω — sequential policy и version fields.

Decoder uses only exact indexed lines through L ; `surface_id` связывает record, но не меняет line-code bindings. Если $D(r) = (prem(r), concl(r))$, definition rule фиксирована даже при $r \notin E$. Decoder δ — одна deterministic partial function во всём core; скрытая смесь function и relation запрещена.

2.3. Domain gates

Contract well-formed, только если:

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

и для каждого $r \in R$:

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

Unknown code, duplicate protected id, missing rule definition или target вне Q ставят input вне valid checker domain. Top token `VALID-PROOF-READSET-ANALYSIS` для такого input не выдаётся.

Слова `standard`, `formal` или `obvious` в payload не проходят эти gates.

3. Sequential validator

3.1. Decode

Пусть supplied surface содержит exact lines $(\ell_1, \dots, \ell_n)$. Checker идёт слева направо. На достигнутом index i сначала вычисляется $\delta_C(i, \ell_i)$. Если value не определено, contract-local fields:

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

и run немедленно останавливается. Если value определено, фиксируется единственный code

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

Только если каждый reached line декодирован и admitted, traversal достигает следующего index. Поэтому later unmapped line не переопределяет earlier `LINE-NOT-ADMITTED`.

3.2. Step admission

Для каждого reached decoded $i \in \{1, \dots, n\}$:

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

Точное условие:

$$Good_i(C, P) : \Leftrightarrow q_i \in A \vee \exists r \in E : concl(r) = q_i \wedge prem(r) \subseteq H_i.$$

Premises должны находиться раньше conclusion. Future line не поддерживает earlier step. Если least i не удовлетворяет GOOD, contract-local fields:

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

Diagnostic хранит i , q_i и considered enabled rule ids.

3.3. Full admission predicate

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

Если все steps good, но $q_n \neq t$, contract-local fields:

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

Если ADM истинно, contract-local fields:

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

ACCEPTED здесь value поля named contract-result, не standalone claim. Ни один field value не сокращается до bare PROOF или bare TRUE.

3.4. Validator

Пусть j — least reached index, на котором либо $\delta_C(j, \ell_j)$ undefined, либо decoded q_j не удовлетворяет $Good_j$. Тогда exact sequential status:

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

Поле $\tau_C(P)$, записанное runtime path terminal_witness.kind, хранится отдельно как diagnostic причины или terminal gate. Оно не является второй компонентой $V_C(P)$ и не участвует в классификации PSR-5 divergence. $V_C(P)$ — code отношения между P и C . Он не предъявляет referent claim-codes.

4. Extensional ReadSet

4.1. Sufficient set

Пусть input record имеет coordinates $x = (x_1, \dots, x_m)$. $S \subseteq \{1, \dots, m\}$ sufficient для F , если:

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

4.2. Necessary coordinate

Coordinate k necessary extensionally, если существует collision pair x, y :

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

Каждый sufficient ReadSet обязан различать такую pair. ReadSet здесь не психологическое утверждение о том, что checker «видит» поле; это extensional necessity, заданная equality и output collision.

4.3. Contract projection

Пусть k_C — index contract projection в input tuple, а $proj_C(x) = C$. Общая PSR-5 утверждает necessity k_C , не membership самого record C в множестве indices.

Для усиленной minimal pair вводится coordinate

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

В runtime schema coordinate E записана field `enabled_rule_ids`.

Semantic erasure удаляет diagnostic id и enabled field:

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

Именно эта projection должна совпасть в minimal pair; только E различается.

`enabled_rule_ids`

Все остальные semantic coordinates фиксированы extensionally. Diagnostic `reading_contract_id` может различаться, но не входит в semantic validator projection.

5. PSR-5

5.1. Общая формулировка

Пусть exact P fixed и C_1, C_2 принадлежат declared finite contract family C . Если:

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

то contract projection, различающая C_1, C_2 , necessary для status на C .

Здесь и в PSR-5 $V_C(P)$ — только значение поля `proof_status`. Поэтому PSR-PREMISE истинна ровно при разных `proof_status`; отдельные $\tau_{C_1}(P), \tau_{C_2}(P)$ не сравниваются классификатором divergence.

$$V_{C_1}(P) \neq V_{C_2}(P) \Rightarrow k_C \in ReadSet_C(V)$$

Box означает extensional necessity $proj_C$, не необходимость каждого unused bit. Для minimal pair усиление имеет вид:

$$k_E \in ReadSet_{\{C_1, C_2\}}(V).$$

5.2. Эквивалентная невозможность

Не существует core-surface-only функции $F: P \rightarrow S_V$, которая для одного P , не получая contract difference, возвращает оба отличных required statuses:

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

5.3. Доказательство

Предположим такую F . В обеих instances её argument равен exact P , следовательно, output $F(P)$ один. Но PSR-PREMISE требует два разных outputs. Противоречие. Если sufficient ReadSet не различает C_1, C_2 , pair совпадает на read coordinates, но outputs различаются, что нарушает SUFFICIENT. Следовательно, различающая contract projection necessary. ▢

5.4. Область квантора

PSR-5 квантифицирует по exhibited C , а не по всей математике. Фраза вся математика не задаёт domain своим появлением. Результат достаточно силен без historical origin claim: proof-status в declared family сейчас не является функцией lines alone.

6. Минимальная collision pair

6.1. Fixed surface

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

Читаемая печать:

```
a
a → b
b
```

Exact canonical surface_sha256 h_{ab} одинаков в обеих instances; raw container JSON bytes не сравниваются.

6.2. Fixed coordinates

В обоих semantic contracts совпадают:

- exact line lexicon;
- deterministic parser;
- code universe $Q = \{A, I_{AB}, B\}$;

- rule universe $R = \{m p\}$;
- assumptions $A_0 = \{A, I_{AB}\}$;
- full ground-rule definition;
- target $t = B$;
- sequential policy;
- inert payload;
- validator version.

Diagnostic `reading_contract_id` может различаться только как `ignored label`. После удаления id_C и `enabled field`:

$$erase_{id,E}(C_1) = erase_{id,E}(C_2).$$

Per-line decoder фиксирован; tuple ниже является shorthand трёх indexed applications:

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

Ground-rule definition фиксирована:

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

6.3. Единственное различие

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

Semantic difference ровно одна:

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

То есть `enabled_rule_ids` отличается одним id; diagnostic contract ids исключены из validator semantics.

6.4. Первый output

В C_1 :

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

поскольку $m p \in E_1$, его conclusion B , а premises находятся в H_3 . Итог:

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

6.5. Второй output

В C_2 , первые два steps assumptions, но:

$$Good_3 = 0$$

поскольку $E_2 = \emptyset$. Итог:

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

6.6. Усиленный вывод

Не только P , но также lexicon, parser, codes, assumptions, full rule definitions, target, payload и validator fixed. Output collision вызывается единственным enabled-set bit. Следовательно, enabled_rule_ids necessary coordinate для этой exhibited family. Rule definition, записанная, но выключенная, не становится enabled от своего текста.

7. ADM-5

7.1. Admission inputs

GOOD показывает exact sources admission:

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

Per-line decoding задаётся defined values $\delta_C(i, \ell_i)$ на reached indices. Target admission задаётся $q_n = t$ после полного successful traversal.

7.2. Finite typed sequential witness

Witness имеет вид

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

Каждый admission node типизирован одним из constructors:

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

или

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

Для RULE-node $k \geq 1$: runtime model не допускает ground rule с пустым множеством premises.

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

1. для каждого i : $q_i = \delta_C(i, \ell_i)$;

2. для ASSUMPTION-node $q_i \in A$;
3. для RULE-node $r \in E$, все $j_m < i$, $concl(r) = q_i$;
4. set claim-codes at cited positions равен $prem(r)$;
5. $q_n = t$.

Base-free labels, prose justifications и untyped pointers не являются witness nodes.

7.3. Формулировка ADM-5

$$ADM_C(P) \Leftrightarrow \exists W \text{ finite typed sequential} : Cert_C(P, W)$$

7.4. Доказательство

Пусть $ADM_C(P)$. Для каждого good step выбирается один finite admission source: ASSUMPTION при $q_i \in A$, иначе enabled rule r из GOOD. Для каждой premise выбирается earlier position с нужным code. Конечность sequence даёт finite W ; target condition даёт final gate.

Обратно, пусть certificate verifier принимает W . Каждый ASSUMPTION-node удовлетворяет первой disjunct GOOD, каждый RULE-node — второй. Decode, $n > 0$ и $q_n = t$ проверены. Следовательно, $ADM_C(P)$. $\square$

ADM-5 является positive certificate/witness equivalence. Она не получает силу от слова certificate: verifier conditions записаны отдельно.

7.5. Payload noninterference corollary

Пусть M — inert payload вне canonical P, C . При fixed protected fields:

$$ADM_{C,M}(P) = ADM_{C,M'}(P).$$

Например, M_0 empty, а M_1 содержит:

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

При одинаковых P_{ab}, C_{NO_MP} оба records rejected unsupported step. Self-label находится только в inert M_1 ; P, C , decoder output и certificate search неизменны.

Строка AXIOM не вносит code в A ; строка RULE не вносит id в E ; строка PROOF не делает $q_n = t$.

7.6. Declaration constructor как non-executable extension

Frozen runtime Phase 5 не имеет constructor, который мутирует axiom_codes по prefix surface-line: assumption set фиксирован до run. В явно расширенном model contract мог бы задать policy: line c

prefix АИОМ: декодируется и добавляется в assumptions. Тогда constructor сам являлся бы новой protected coordinate ω . Это theorem-only extension, не behavior опубликованного checker. Даже в extension pair (P, C) , а не P alone, определяет effect declaration.

8. Дополнительные exact counterexamples

8.1. Parser pair

Одна raw line $x * y$ может decode как $MUL(X, Y)$ под δ_1 и как $LITERAL_STAR$ под δ_2 . При fixed surface и different outputs decoder projection necessary. Эта pair общая, но слабее minimal pair: в ней меняется decoder.

8.2. Assumption pair

Surface из одной line p . При $Pcode \in A_1$ и $Pcode \notin A_2$, остальных fixed fields, statuses differ. Raw p не содержит assumption membership.

8.3. Target pair

All steps admitted, final code B . Contracts differ only $t_1=B$, $t_2=C$. Output collision делает target coordinate necessary. Final line не выбирает target автономно.

8.4. Order control

Surfaces $a / a \rightarrow b / b$ и $b / a / a \rightarrow b$ различны. Это surface sensitivity, не PSR collision при fixed P . Контроль нужен, чтобы не приписывать contract theorem изменению exact line sequence.

9. Meta-proof

9.1. Lift

Пусть M названо metaproof admission контракта C . Для проверки требуется meta-contract D :

$$W_D(M, C).$$

D задаёт meta-parser, meta-assumptions, meta-rules и meta-target.

9.2. Meta collision

Если при fixed (M, C) :

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

то D necessary по той же collision argument. Meta-proof переносит вопрос в новый contract. Он не создаёт contract-free admission.

9.3. Конечная остановка

Никакого тезиса о mandatory infinite regress нет. Procedure может остановиться на explicitly declared D . Но boundary принимаем D остаётся input. Строка D validates itself не удаляет D из W_D .

10. Gates

Gate 0 — EXACT-SURFACE

- surface_id и exact lines представлены;
- no silent scalar normalization;
- surface_sha256 записан;
- collision pair сохраняет P .

Gate 1 — CONTRACT-SHAPE

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

Gate 2 — DETERMINISTIC-DECODE

- exact line lexicon fixed;
- δ deterministic partial function indexed exact line-records;
- every reached supplied line accounted for, включая reached line с line_text == "";
- first reached undefined decode получает proof_status: UNPARSED; earlier terminal rejection wins over any later line.

Gate 3 — SEQUENTIAL-GOOD

- every q_i satisfies GOOD;
- rule id enabled;
- premises earlier;
- self-label не shortcut.

Gate 4 — TARGET

- $n > 0$;
- $q_n = t$;
- target принадлежит declared universe.

Gate 5 — PROVENANCE

- artifact_sha256, surface_sha256 и per-contract derivation_core_sha256 записаны;
- admission-kind каждого step записан;
- enabled id и premise indices recorded;
- validator version fixed.

Failure gate исключает acceptance только under named contract. Он не создаёт bare negative или world status.

11. Certificate

Executable output использует только 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 ИЛИ 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, не machine tokens: ACCEPTED означает ADM true under named contract; REJECTED — first reached mapped line failed admission либо full traversal failed target gate; UNPARSED — first reached line lacked exact mapping under named contract.

Replay exact, только если:

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

Здесь h_A, h_S, h_D обозначают runtime fields artifact_sha256, surface_sha256, derivation_core_sha256 соответственно.

Trace и terminal witness также должны совпасть. Certificate показывает recorded dependency; он не предъявляет referent claim-code.

Это replay requirement, а не PSR classification rule: pairwise PSR divergence вычисляется только по неравенству `proof_status`; `terminal_witness.kind` остаётся отдельной diagnostic coordinate.

12. Запрещённые экспорты

Из

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

не следуют без новых coordinates:

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

Запрещены сокращения:

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

Следующий файл Phase 5B вводит отдельный export binding I , не скрывая его в слове proof.

13. Extensions

13.1. Nondeterministic output

Если validator возвращает set $V_C(P) \subseteq S_V$, PSR применяется к different output sets. Contract coordinate остаётся necessary.

13.2. Probabilistic output

Если output — distribution $\mu_{C,P}$, то $\mu_{C_1,P} \neq \mu_{C_2,P}$ создаёт collision. Probability не стирает ReadSet.

13.3. Interactive run

Transcript T входит в input $V_C(P, T)$. PSR требует fixed (P, T) ; изменение T — другая pair.

13.4. Executable package

Kernel, imports, environment и flags входят в расширенный C^+ . Status остаётся ...UNDER-DECLARED-CONTRACT. Слово assistant не предъявляет внешнего проверяющего.

14. Limits

PSR-5 устанавливает necessity coordinate в exhibited family. Она не ранжирует contracts и не утверждает, что accepted derivations бесполезны. Она не утверждает, что computation не может

изменять устройство. Изменённое устройство будет отдельным presented record, но не referent слова автоматически. Она не устанавливает historical origin математики, состояние человека, universal impossibility of binding или world falsity. ADM-5 не запрещает declaration constructor. Она требует показать его в protected C . Название proof, формула PSR-5 и знак $\square$ не получают authority от себя.

15. Falsifiers

15.1. PSR falsifier

Для снятия PSR collision надо предъявить $F(P)$, которая при одном exact P , не получая различие C_1, C_2 , возвращает оба distinct deterministic statuses в одном output namespace. Если F возвращает pair или set, задача изменена: contract-indexed statuses упакованы, а не получены из P alone.

15.2. Pair defeaters

Применение не проходит, если:

- surfaces различны;
- outputs совпадают;
- меняется coordinate кроме заявленной;
- один contract вне declared C ;
- сравниваются разные status projections;
- contract hidden inside object, названного P .

15.3. ADM falsifier

Если only-untrusted metadata меняет output при fixed P, C , checker нарушает PAYLOAD-NONINTERFERENCE. Это обнаруживает hidden input, не self-authority metadata.

16. Самограница

Этот Markdown-файл — exact surface P_{PSR} . Строки PSR-5, теорема и $\square$ входят в неё. Для validation требуется C_{PSR} : parser Markdown/math, assumption set, ground rules, enabled ids и target. Другой contract может вернуть другой scoped status. Это не исключение; это собственная instance PSR-5. Файл не предъявляет автора, читающего, математику, истину или мир. Он предъявляет lines и declared relations между codes.

17. Итоговый нож Phase 5A

Сначала записаны lines. Чтобы получить derivation status, отдельно вводятся parser, assumptions, ground rules, enabled ids и target. Минимальная pair фиксирует всё, включая full rule definitions и payload. Меняется только:

enabled_rule_ids

и acceptance меняется.

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

Self-label не заменяет C . Meta-proof вводит D , а не стирает contract-coordinate.

Contract-local proof_status: ACCEPTED фиксируем status прохождения declared gates. Отдельный diagnostic terminal_witness.kind: TARGET-ACCEPTED записываем terminal gate и не классифицируем PSR-5 divergence. Ни одно из этих полей не предъявляет доказанное, referent, истину, источник, применение, цель или смысл.

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

Последняя строка также остаётся строкой.

02. ТЕОРЕМА ОТСУТСТВИЯ АВТОНОМНОГО ЭКСПОРТА

ДОПУЩЕННЫЙ ВЫВОД НЕ ПРЕДЪЯВЛЯЕТ ПРИМЕНЕНИЕ

0. Область результата

Этот файл продолжает Phase 5A после scoped status:

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

Принимается fixed derivation core (P, C) . Вопрос Phase 5B: достаточно ли accepted core, чтобы автономно получить application-code, goal-code или external-claim-code? Ответ NEX-5 относится к declared finite binding family. Он не квантифицирует по всем применениям и не устанавливает мировой отрицательный тезис. Строки

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

остаются строками. Они не предъявляют ракету, Марс, цель, мир или применение.

1. Fixed accepted derivation core

1.1. Core

Пусть $P = (surface_id, lines)$ — exact canonical surface record и C — finite typed reading contract Phase 5A. Предполагается:

$$ADM_C(P) = 1$$

и contract-local fields:

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

Selected target-code:

$$t = t_C \in Q_C.$$

Acceptance остаётся relation (P, C) . Она не превращает t в referent.

1.2. Derivation core serialization

Canonical derivation core включает ровно:

- surface_sha256 canonical {surface_id, lines};

- full ordered decoding records;
- axiom_codes;
- full normalized rules[`{rule_id,premise_codes,conclusion_code}`] catalog, без per-rule payload;
- enabled_rule_ids;
- target_code.

Checker строит именно named JSON object, не tuple или array:

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

И определяет:

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

где D — этот named object. Порядок object keys в исходной сериализации не является coordinate: canonicalizer задаёт canonical key order. Порядок members внутри protected arrays остаётся частью object value. Runtime field name:

derivation_core_sha256

1.3. Что core digest исключает

d не включает:

- export binding I ;
- outer descriptions;
- application labels;
- goal labels;
- claim labels;
- package filename;
- untrusted metadata;
- trace и terminal witness;
- diagnostic reading_contract_id;
- raw container JSON bytes. Это исключение обязательно для collision pair: если I включить в сравниваемый digest, fixed-core premise будет скрыто нарушена. Full artifact hashes при $I_1 \neq I_2$ ожидаемо различаются. NEX-5 сравнивает одинаковый derivation_core_sha256, а не full

artifact hash. Digest остаётся code bytes. Он не предъявляет derivation, referent или источник своим названием.

2. Opaque export sorts

2.1. Разделённые namespaces

Фиксируются конечные tagged namespaces:

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

Здесь:

- Q — internal claim-codes;
- U — application-codes;
- G — goal-codes;
- W — external-claim-codes. Слово `external` означает только отдельный namespace. Оно не предъявляет внешнее.

2.2. Opaque-code discipline

Нормативные examples используют codes:

APP-07
GOAL-03
CLAIM-11

Их читаемые labels находятся в optional outer description, а не в identity code. Даже label `rocket` не является `rocket`. Даже tag `GOAL` не является целью.

2.3. No cross-sort coercion

Без explicit constructor запрещены conversions:

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

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

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

Одинаковая character sequence в разных tagged namespaces не снимает type boundary.

3. Export binding I

3.1. Typed record

Binding record:

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

U, G, W — declared finite universes. v_I — canonicalization и version policy.

B_I — finite partial map:

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

где D — namespace derivation-core digests. Аргументом lookup является exact pair (d, t) .

3.2. Binding entry

Entry

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

устанавливает только recorded association между four codes. Она не устанавливает:

- что u имеет referent;
- что g является чьей-либо целью;
- что w истинно;
- что named application выполнено;
- что core причинно создал внешний объект;
- что слова outer description соответствуют чему-либо вне record.

3.3. Well-formedness

I well-formed, только если:

- all digest keys canonical;
- all internal codes belong to Q ;
- all outputs typed in $U \times G \times W$;
- duplicate key has exactly one output;
- no implicit default binding;
- no wildcard outside explicit policy. В theorem-only model violation получает exposition diagnosis:

INVALID-EXPORT-BINDING-RECORD

Это не token executable checker verdict schema. Слово official в payload не проходит well-formedness.

4. Abstract export function

Разделы 2–10 задают theorem-only finite model. Labels BOUND-EXPORT-UNDER-DECLARED-BINDING, UNBOUND-EXPORT-UNDER-DECLARED-BINDING и INELIGIBLE-CORE-FOR-EXPORT-UNDER-DECLARED-CONTRACT ниже являются abstract values X_I , не machine tokens executable checker.

4.1. Lookup

Для fixed accepted core (d, t) :

$$X_I(d, t) = \begin{cases} (BOUND - EXPORT - UNDER - DECLARED - BINDING, u, g, w), & B_I(d, t) = (u, g, w), \\ (UNBOUND - EXPORT - UNDER - DECLARED - BINDING, \perp, \perp, \perp), & (d, t) \notin \text{dom}(B_I). \end{cases}$$

Abstract value UNBOUND... означает lookup nonmembership только under exact I . Он не сокращается до NO APPLICATION, NO GOAL, FALSE или NO REFERENT.

4.2. Core acceptance gate

Если $ADM_C(P) = 0$, abstract function не выполняет lookup как accepted derivation export и возвращает theorem-only value:

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

Это status pipeline, не мировое утверждение.

4.3. Determinism

В normative core B_I — deterministic partial function. Multi-binding relation рассматривается отдельно в extensions. Смешивать function и relation в одном theorem instance запрещено.

5. Extensional export ReadSet

5.1. Collision definition

Пусть k_I — index binding coordinate, $\text{proj}_I(O) = I$. Coordinate k_I necessary для X на family I , если существуют $I_1, I_2 \in I$:

$$\begin{aligned} d_1 &= d_2 = d, t_1 = t_2 = t, \\ I_1 &\neq I_2, X_{I_1}(d, t) \neq X_{I_2}(d, t). \end{aligned}$$

Это extensional definition. Оно не утверждает, что программа психологически читает binding.

5.2. Outer records

Пусть full outer artifact:

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

где Z_i — opaque descriptions и metadata. Collision pair требует equality derivation core:

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

Full artifact equality не требуется и обычно ложна, потому что $I_1 \neq I_2$. Descriptions Z_i не входят в binding lookup, если explicit policy не делает их protected input.

6. NEX-5

6.1. Формулировка

Пусть (P, C) accepted under declared contract, d — fixed core digest, t — fixed target и $I_1, I_2 \in I$. Если:

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

то coordinate k_I , чья projection есть $proj_I$, necessary для export status/value на declared family I .

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

Следовательно, accepted (P, C) alone не определяет оба export outputs.

6.2. Невозможность core-only function

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

при distinct required outputs и отсутствии I или его encoded equivalent в arguments.

6.3. Доказательство

Предположим core-only function $F(d, t)$. Collision pair имеет одни d, t , поэтому function output один. NEX-PREMISE требует разные outputs. Противоречие. ReadSet, не различающий I_1, I_2 , видит одинаковые retained coordinates, но функция X различается. Такой set не sufficient. Значит, distinguishing binding projection necessary. ▢

6.4. Scope

NEX-5 относится к exhibited I . Она не утверждает, что никакое применение невозможно. Она утверждает точное: different exports при fixed accepted core требуют coordinate, где записано различие. Фраза реальное применение не расширяет domain и не предъявляет применение.

7. Минимальная binding pair

7.1. Fixed core

Берётся accepted P_{ab}, C_{MP} Phase 5A:

a
a → b
b

Bce derivation protected fields fixed. Target $t = B$. Core digest d_{ab} fixed.

7.2. Fixed universes

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

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

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

Universes, canonicalization и version policy 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

Surface, contract, admission certificate, target и derivation core digest equal. Only binding entries differ. Export values differ. По NEX-5 I necessary. Ни P , ни scored acceptance не выбирают APP-07 вместо APP-08.

7.6. Outer descriptions control

Z_1 может содержать labels ракета, Марс, цель полёта. Z_2 может содержать другие labels. Эти strings не участвуют в minimal collision proof. Даже при наличии они не предъявляют названное.

8. Unbound не является отрицательным мировым status

8.1. Empty binding

Пусть:

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

Тогда:

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

8.2. Запрещённый переход

Не допускается:

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

Также не допускаются:

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

8.3. Причина

Lookup установил лишь nonmembership key в finite map B_I . Никакого world-state predicate в model нет. Чтобы получить negative status в другом domain, потребуется отдельный finite contract, provenance и completeness boundary. Молчание binding record не создаёт их.

8.4. Missing I

Если binding record вообще не предъявлен, executable fields:

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

Пустые arrays сообщают только нулевое число explicit binding records в input package. Они не сообщают отсутствие применения, цели или referent.

9. Записанный binding не предъявляет referent

9.1. Record-of-association

Даже well-formed entry

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

предъявляет codes и membership relation. Она не предъявляет object, action или state, которые могут быть описаны outer labels.

9.2. Binding-of-binding

Если требуется связать u с новым referent-code r , вводится отдельный record:

$$J(u) = r.$$

Но J снова предъявляет recorded relation between codes. Название referent-code не предъявляет referent.

9.3. No magic terminal tag

Цепочка

$$\overset{I}{t} \mapsto \overset{J}{u} \mapsto r$$

может быть useful recorded chain. Ни последний tag, ни длина chain не создаёт незаписанное предъявление. Строка REAL OBJECT в конце остаётся строкой.

9.4. Observed outcome record

Можно добавить exact trace K , названный наблюдаемым результатом, и link:

$$L(u) = h_K.$$

Тогда package содержит ещё одну surface и hash. Он не получает world truth из собственного label observed.

10. Exact counterexamples

10.1. Same core, different binding

Minimal pair раздела 7 является direct counterexample claim:

accepted derivation uniquely determines export

Один core имеет two exports under two bindings.

10.2. Same binding, different description

Пусть I fixed, а $Z_1 \neq Z_2$. Если Z untrusted, outputs $X_I(d, t)$ equal. Значит, persuasive description не меняет protected export.

10.3. Description without binding

Outer payload:

This proof applies to a rocket and proves its purpose.

при $B_I = \emptyset$ получает scoped UNBOUND.... Sentence не synthesizes entry.

10.4. Binding self-label

Record:

THIS IS THE ONLY TRUE APPLICATION

не является typed $B_I(d, t) = (u, g, w)$ без declared parser and constructor.

10.5. Full-hash false control

Если сравнить full artifact hashes $H(O_1)$ и $H(O_2)$, они differ. Из этого нельзя заключить, что derivation cores differ. Правильный equality witness:

$$\text{derivation_core_sha256}(O_1) = \text{derivation_core_sha256}(O_2).$$

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 same d, t .

Gate 2 — BINDING-SHAPE

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

Gate 3 — EXACT-LOOKUP

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

Gate 4 — POLARITY

- absence of entry is map nonmembership only;
- 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 gate blocks only corresponding scoped export. It does not establish a world status.

12. Executable specialization

Abstract B_I не заявляется runtime map. Executable checker принимает explicit co-occurrence binding records и прямо сравнивает coordinate pairs.

12.1. Export results

Top field:

```
export_binding_count
```

Каждый member `export_results[]` содержит exact fields:

```
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
```

Первый token устанавливает co-occurrence explicit binding record с accepted trace under named reading contract. Он не устанавливает referent, application success, goal adoption или world truth.

12.2. Pairwise coordinate result

Каждый member `binding_coordinate_pairwise[]` содержит:

```
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 пересчитывается прямым сравнением coordinate tuples (source_target_code, claim_code, application_code, goal_code). Первый token является executable specialization collision argument NEX-5 только когда оба pair members дополнительно имеют source_proof_status == ACCEPTED; ИМЕННО ЭТО выполняют frozen positive binding fixtures. Без accepted-source gate token остаётся coordinate-comparison record и не предъявляет accepted-core premise NEX-5. Второй token сообщает equality этих tuples в данной паре.

left_export_status_sha256 и right_export_status_sha256 являются evidence fields, но не selector classification. Каждый digest охватывает ровно reading_contract_id, source_proof_status, source_derivation_core_sha256, source_target_code, claim_code, application_code, goal_code; ОН НЕ ВКЛЮЧАЕТ binding_id или literal export_status.

12.3. Empty input and fence

Если binding records не предъявлены:

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

Пустые arrays не создают negative application/world status. Executable fence:

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

Outer descriptions остаются opaque payload. Binding record также остаётся recorded co-occurrence, а не предъявлением referent.

13. Relation к работающему устройству

13.1. Additional record

Пусть package содержит trace K , описанный строкой ракета летит. Trace может быть bytes, measurements, images или event codes. Все они являются additional presented records. Их наличие не меняет P, C и не создаёт binding I без protected constructor.

13.2. Application success

Можно определить новый contract C_{succ} , predicate $Succ_{C_{succ}}(K)$ и scoped status:

```
SUCCESS-UNDER-DECLARED-APPLICATION-CONTRACT
```

Это новый readset (K, C_{succ}) . Он не превращается в bare SUCCESS.

13.3. Для чего

Goal-code g может быть связан с application-code u . Этот binding устанавливает only code relation. Вопрос для чего? входит как written question. Answer-code входит как written answer. Ни один не получает meaning от своей грамматической формы. Даже success under C_{succ} не выводит goal или meaning без declared binding.

14. Extensions

14.1. Multi-binding relation

Если B_I relation, output становится finite set:

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

Different sets при fixed core снова делают I necessary.

14.2. Prioritized binding

Если multiple entries разрешены, priority order входит в I . Смена order при смене output создаёт отдельную necessary coordinate.

14.3. Dynamic registry

Timestamp, registry snapshot и resolver version входят в expanded I^+ . Слово current не фиксирует snapshot.

14.4. Probabilistic export

Для distribution $\mu_{I,d,t}$ different distributions при fixed core дают NEX collision. Probability не предъявляет referent.

14.5. Human selection record

Selection by annotator represented codes (a, η) . Они входят в binding readset, если влияют на output. Модель не утверждает состояние названного человека.

15. Limits

NEX-5 не утверждает:

- что application невозможно;
- что устройства не работают;
- что goals нельзя записывать;
- что bindings бесполезны;
- что все bindings equally admissible;

- что ни у одного code нет referent;
- что мир отсутствует;
- что математика исторически произошла из языка;
- что proof systems ничего не вычисляют. NEX-5 утверждает necessity I на exhibited collision family и запрещает скрытый export из accepted core alone. Тезис recorded binding не предъявляет referent является boundary предъявленной формы: shown entry есть lines/codes. Он не получает referent certification от tag.

16. Falsifiers

16.1. NEX falsifier

Нужно предъявить core-only $F(d, t)$, которая при fixed d, t , не получая I_1/I_2 или encoded equivalent, возвращает оба distinct required deterministic outputs в одном namespace. Pair/set output меняет codomain и упаковывает alternatives; он не выбирает оба как один value.

16.2. Collision defeaters

NEX application снята, если:

- derivation core digests differ;
- target differs;
- core acceptance differs;
- outputs equal;
- binding difference hidden in d ;
- one I_i outside declared family;
- full artifact hash ошибочно использован вместо core digest.

16.3. Negative-default falsifier

Чтобы получить negative world status из unbound lookup, надо добавить explicit world-status contract и completeness certificate. Это будет expanded input, а не вывод из missing I .

16.4. Referent-presentation test

Если заявлено, что entry сама предъявляет referent, требуется показать, где в exact record находится не новый code/description, а названное. Добавление ещё одного label переносит запись; оно не снимает boundary.

17. Самограница

Этот файл — exact surface P_{NEX} . Он содержит слова export, application, goal, referent и NEX-5. Они не предъявляют названное. Формулы требуют reading contract, а theorem status требует declared

metatheory. Если файл связан на сайте с label исследование, link и label создают outer record. Они не создают authority. NEX-5 применима к себе: разные outer bindings этого файла при fixed core требуют coordinate I . Самоприменение не расширяет квантор за declared family.

18. Итоговый нож Phase 5B

Accepted derivation core фиксирует:

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

Export имеет другую форму:

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

Minimal pair сохраняет d, t , но меняет $proj_I$, и export меняется. Поэтому k_I necessary. Если explicit binding records отсутствуют, executable result:

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

а не отсутствие применения, referent, цели или мира. Если I присутствует, он предъявляет recorded association between opaque codes. Он не предъявляет referent собственным tag.

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

03. АКСИОМА И ПРАВИЛО НЕ ДОПУСКАЮТ СЕБЯ

Координаты parser, checker, compiler и meta-proof в статусе доказательства

Phase 5 • техническое исследование • 29 августа 2026

Кирилл Шебетов • beforeword / research

Сначала предъявлена строка.

Затем предъявлено правило её допуска.

Правило не обнаруживает свой допуск внутри собственной строки.

Аннотация

Этот файл разбирает слово доказательство до точных координат допуска. Различаются:

- байтовая запись;
- tokenization;
- parser;
- grammar;
- axiom admission;
- rule admission;
- proof-checker;
- compiler;
- runtime;
- meta-proof;
- payload labels;
- application export.

Каждый статус вводится только явно записанным отношением. Ни строка аксиома, ни строка правило, ни строка валидно не выполняют это отношение. Основной результат **ADR-5** — ADMISSION DEPENDENCY READSET — устанавливает readset-необходимость: если одна и та же exact запись P получает разные статусы при двух admissible contracts, различающая contract-coordinate входит в полный readset статуса. Статус не является функцией только от P . Результат **NSA-5** локализует начало допуска: в finite positive nonnull admission system цикл axioms, rules, checkers или meta-proofs без admitted base не создаёт ни одного admitted node в least closure. Бесконечный регресс также не становится

началом допуска. Он лишь записывает ещё одно требование допуска. Inconsistent calculus и explosion принимаются как положительные controls. Если contradiction-code и explosion-rule допущены, checker может принять каждый formula-code своего exact carrier. Это устанавливает поведение calculus under contract. Оно не превращает принятую строку в названный объект, не вводит внешнюю истинность и не предъявляет цель.

PREDECESSORS-AS-PREMISES: NONE.

0. Сначала предъявлены записи

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

До введения exact contract это строки. Строка axiom не допускает соседнюю строку как аксиому. Строка rule не разрешает переход. Строка parser не разбирает себя. Строка proof не предъявляет proof-status. Строка valid не выполняет проверку. Строка sound не экспортирует результат. Строка checker accepted может находиться внутри payload отвергнутого объекта. Строка runtime returned 0 может быть нечитанным комментарием. Строка meta-proof complete не является meta-proof witness. Строка therefore true добавляет знаки. Она не добавляет названное. Нулевая граница:

payload label ≠ protected status field.

Даже protected status field требует producer, schema, input binding и acceptance contract.

Соглашение о labels: все uppercase statuses/predicates в этом файле являются exposition/reader-gloss, кроме явно названных exact checker enums/fences. В частности, VALID-DERIVATION-IN-C, CHECKER-ACCEPTED-UNDER-HASHED-CONTRACT, ACCEPTED-PROOF-UNDER-DECLARED-CONTRACT, ADMITTED-AXIOM-IN-C, а также table labels TOKENIZED/PARSED/AXIOM/RULE/BUILT/RAN/META-ADMITTED не являются значениями verdict schema. Exact runtime proof_status принимает только ACCEPTED, REJECTED или UNPARSED; exact terminal kinds и overall/fence tokens перечисляются там, где они используются.

1. Exact носители

Фиксируются конечные попарно типизированные носители:

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 .

Совпавшие печатные строки не объединяют sorts. Formula-code r_1 не становится rule-id r_1 . Byte string a не становится axiom-id a . Runtime trace не становится derivation-code из-за одинакового hash payload. Contract имеет точную форму:

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

Здесь:

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

Компоненты перечислены как input. Их написание не подтверждает их применение. В exact Phase 5 runtime supplied surface уже разделена на indexed lines: τ является identity над line records, а π —

partial exact line-text/code decoding из frozen lexicon. Более общий tokenizer/parser notation этого файла служит explanatory supertype и не расширяет executable theorem-domain ADM-5.

2. Защищённые статусы

Для $p \in B$ определяются отдельные 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 здесь является только serialization interface witness. Exact runtime нормализует каждую rule definition в ground record (rule_id, premise_codes, conclusion_code); порядок premises не создаёт дополнительного допуска, substitutions или schema-level metavariables нет. Generated trace step имеет другой exact shape: line_index, statement_code, admission_kind, rule_id, premise_witnesses. Certificate equivalence ADM-5 из файла 01 применяется к этому exact ground checker; ADR-5 ниже является отдельным 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:

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

Ни один predicate не сокращается до наличия соответствующего слова в p . В частности:

$$\text{VALID} \sqsubseteq p \neq \text{VD}_C(p, f).$$

И:

$$\text{AXIOM} \sqsubseteq p \neq \text{AX}_C(a, f).$$

3. ADR-5: ADMISSION DEPENDENCY READSET

Пусть C — объявленный класс contracts. Пусть protected public output использует exact runtime enum:

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

где:

- **ACCEPTED** означает: ground trace завершён TARGET-ACCEPTED under exact contract;
- **REJECTED** означает: parsed trace не прошёл line-admission либо target gate;
- **UNPARSED** означает: на first reached terminal index отсутствует exact decoding; earlier admission failure уже дала бы REJECTED и остановила traversal.

Эти три строки являются scoped runtime codes. **ACCEPTED** не переименовывается в **TRUE**, **REJECTED** не переименовывается в **FALSE**, а **UNPARSED** не предъявляет отсутствие объекта.

ADR-5

Если существуют $p \in B$ и $C_0, C_1 \in C$, такие что

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

то не существует функции

$$v : B \rightarrow V$$

с условием

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

Следовательно,

$$\text{status}(p) \neq \text{status}(p, C)$$

как взаимозаменяемые signatures.

Доказательство

Предположим, что такая v существует. Тогда

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

и

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

По транзитивности равенства:

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

Это противоречит исходному различию outputs. Значит, exact bytes p недостаточны для обоих результатов. Различающая coordinate должна быть прочитана. ▢ Теорема не говорит, что contracts «нереальны». Она говорит только:

status admission не содержится автономно в неизменившейся строке.

4. Readset-форма ADR-5

Пусть contract разложен:

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

Для deterministic procedure F coordinate c_j входит в extensional readset, если существуют contracts C, C' , различающиеся только по c_j , и input p , для которых

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

Тогда обозначается:

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

RS-5

Если C, C' различаются только по c_j и output различается, то

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

Это определение превращает риторическое «контекст важен» в exact dependency claim. Ни слово контекст, ни слово зависит больше не несут доказательство. Необходим точный witness pair.

5. Strict-prefix witness при неизменных линиях

Exact payload:

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

Оба contracts содержат один и тот же frozen core:

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

Единственная различающая coordinate:

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

Contract C_{MP} задаёт:

- 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 и schema;
- enabled set $\{r_{MP}\}$, записанный runtime field `enabled_rule_ids`;

Тогда:

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

Contract C_{NO-MP} сохраняет те же bytes, surface, lexicon, grammar, axiom table, полный rule catalog, target и checker version. Меняется только protected `enabled_rule_ids`: rule-id r_{MP} остаётся в catalog, но не входит в enabled set. Тогда:

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

Contract C_{GLYPH} читает $\rightarrow$ как inert glyph pair. В этом отдельно объявленном contract exact decoding не строит derivation input:

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

Здесь нет развилки статуса. Первые две строки образуют один и тот же accepted strict prefix в обоих основных contracts. Различается только допуск последнего перехода к b . Байты не изменились. Слово b не стало менее похожим на b . Изменилась одна coordinate допуска. Поэтому `enabled_rule_ids` входит в readset. `VALID-DERIVATION-IN-C` при этом не переименовывается в `TRUE`, а `REJECTED-BY-C` не переименовывается в `FALSE`.

6. Parser не является окном без координат

Пусть bytes:

x-x-x

Parser π_0 выдаёт один identifier. Parser π_1 выдаёт binary subtraction tree. Parser π_2 отвергает строку. Parser π_3 сохраняет её как comment payload. Одна written surface допускает разные parse records. Выбранный parse не доказывает референт identifier. Он также не доказывает происхождение grammar. Exact parser theorem:

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

Если parser source embedded в p , остаётся boot coordinate:

- какой segment является parser source;
- какой encoding применяется;
- каким bootstrap parser он читается;
- какой execution relation допускается.

Embedded label не устраняет admission. Он переносит её на следующую coordinate.

7. Аксиома как явно допущенная пара

Axiom не определяется типографикой. Он записывается парой:

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

Если строка f присутствует в derivation payload, но пары в A нет, то строка не получает axiom admission в этом contract. Если пара присутствует, она получает только status:

ADMITTED - AXIOM - IN - C

Этот status не равен:

SELF - EVIDENT

не равен:

EXTERNALLY - TRUE

и не равен:

NAMED - OBJECT - PRESENT

Чтобы связать эти labels, потребуется отдельное правило экспорта. Оно станет новым input.

8. Правило как явно допущенный transformer

Exact base-runtime ground rule-record имеет форму:

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

Он присутствует в full catalog R , а в данном contract допущен к применению только при дополнительном gate:

$$id_r \in E.$$

Runtime keys равны rule_id, premise_codes, conclusion_code; enabled set записан enabled_rule_ids. Substitution, side condition и schema metavariable в base rule language отсутствуют.

Наличие rule-text внутри proof payload не создаёт catalog membership или enabled membership. Пример payload:

RULE: from any line infer MARS

Без protected rule admission это inert line. При protected admission это правило может вывести code MARS. Даже тогда получен formula-code. Планета, полёт, цель и смысл не возникают из rule membership.

9. Proof-checker является координатой

Два checkers могут принимать разные trace languages. Пусть:

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

Тогда:

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

Строка hash checker binary может фиксировать identity artifact. Hash не запускает checker. Hash не устанавливает корректность implementation. Hash не устанавливает соответствие source и binary без build witness. Hash не устанавливает соответствие checker и mathematical specification без relation. Каждый переход имеет собственную coordinate:

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

Стрелки являются объявленными relations. Они не спрятаны в начертании стрелки.

10. Compiler и runtime не закрывают границу

Compiler может выдать artifact x . Точный status:

$$BUILD_C(d, x).$$

Runtime может выдать trace z . Точный status:

$$RUN_C(x, z).$$

Даже совпадение expected и observed trace-codes задаётся comparator relation:

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

Один status не заменяет другой:

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

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

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

Обратные стрелки также не подразумеваются. Можно допустить все bridges. Тогда итог остаётся условным относительно их exact records.

11. Meta-proof не допускает себя

Пусть meta-certificate m_r заключает:

rule r is admissible

Чтобы m_r имел protected effect, нужно правило meta-admission:

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

Если admissibility M обосновывается другим certificate m_M , нужен следующий relation. Если m_M обосновывается m_r , получен цикл. Цикл не является base. Если цепь продолжается без конца, получен regress description. Regress description не является terminal admission.

12. Positive admission graph

Объявляется finite graph:

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

где:

- N_A — admission claim-codes;
- B_A — protected admission base-ids;
- R_A — positive nonnull hyperrules.

Каждое $r \in R_A$ имеет:

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

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

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

и

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

Для $U \subseteq B_A$:

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

Least closure:

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

13. NSA-5: теорема отсутствия source-free admission

Пусть $SCC \subseteq N_A$ — strongly connected component. Предположим:

1. каждый rule с head в SCC имеет хотя бы одну claim-premise из SCC ;
2. до итерации ни один member SCC не находится в X_0 .

Тогда:

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

Доказательство

На шаге 0 intersection пуст по условию. Предположим, что он пуст на шаге k . Чтобы на шаге $k+1$ появился head из SCC , должно сработать rule с head в SCC . У этого rule есть premise из SCC . По induction hypothesis она отсутствует в X_k . Rule не срабатывает. Значит, intersection пуст на каждом конечном шаге. При стабилизации он остаётся пуст. ▢ NSA-5 не говорит:

axioms impossible

Она говорит:

positive cycle without admitted introduction introduces nothing in least closure

Начало допуска может быть объявлено base-id. Тогда оно является input.

14. Cycle и regress — разные records, одинаковая граница

Cycle:

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

Без base:

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

C base-activated introduction:

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

получается:

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

Base u не «истинен» из-за роли base. Он объявлен admission coordinate. Regress:

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

не принадлежит finite theorem-domain. Но каждая конечная truncation без terminal introduction имеет пустой least closure. Запись и так далее не предъявляет terminal base. Она сокращает запись цепи.

15. Inconsistent calculus как положительный control

Пусть theory допускает:

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

Пусть contradiction rule допущено:

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

Пусть explosion rule допущено:

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

Здесь F_C — exact finite formula carrier, зафиксированный schema до запуска. Тогда checker может принять derivation каждого $\psi \in F_C$. Это не дефект ADR-5. Это expected positive control. Output зависит от наличия:

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

Удаление explosion rule может сохранить contradiction-code, но прекратить universal derivability. Следовательно:

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

Inconsistent acceptance не превращается в отсутствие. Напротив, обе несовместимые строки положительно присутствуют в calculus.

16. Labels внутри payload инертны

Рассмотрим payload:

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

До schema binding это byte string. После JSON parse это object-code. После schema validation некоторые fields могут стать typed payload. Только protected producer relation может записать status. Строка world_true не получает специального исключения. В этой работе unqualified external status не вводится. Допустимый status имеет score:

VALID-DERIVATION-IN-C

Или:

CHECKER-ACCEPTED-UNDER-HASHED-CONTRACT

Payload может процитировать эти labels. Цитата не выполняет producer relation.

17. Четыре состояния admission и label

Пусть:

- $L = 1$, если payload содержит строку VALID;
- $A = 1$, если protected checker принимает derivation.

Все четыре пары допустимы:

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

$(0, 0)$: label отсутствует, admission отсутствует. $(0, 1)$: checker принимает объект без self-label. $(1, 0)$: объект заявляет VALID, checker отвергает. $(1, 1)$: label присутствует и checker принимает. Поэтому:

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

Только дополнительное restriction может убрать два состояния. Restriction является новой coordinate.

18. Protected readset matrix

Output	Минимально читаемые координаты	Не создаётся автоматически
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, full rule catalog, enabled_rule_ids	внешняя истинность
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, meta-rules, meta-bases	source-free authority

Матрица задаёт dependency expectations. Она не утверждает, что перечисленные inputs предъявлены вне record.

19. Попытки обхода

19.1. «Правило очевидно»

очевидно является дополнительной строкой. Она не заменяет membership в R .

19.2. «Все обязаны принять аксиому»

все обязаны не перечисляет carriers, не задаёт quantifier domain и не выполняет admission.

19.3. «Checker проверен checker-ом»

Появился второй checker relation. Нужен его собственный contract.

19.4. «Meta-proof закрывает вопрос»

Meta-proof может закрыть exact target under meta-contract. Он не допускает meta-contract без coordinate.

19.5. «Работающий compiler доказывает правила»

Compiler output и rule admissibility имеют разные types. Bridge может быть записан. Он не следует из совпадения слов.

19.6. «Практический успех делает proof истинным»

Практический success-code является ещё одним status. Его связь с proof требует application binding.

20. Exact falsifiers

Работа не объявляет себя неуязвимой словом теорема. Для каждого результата записывается falsifier boundary.

20.1. Falsifier ADR-5

Предъявить:

1. exact p, C_0, C_1 ;
2. подтверждённое $V(p, C_0) \neq V(p, C_1)$;
3. функцию $v(p)$, одновременно равную обоим различным outputs.

Такой объект нарушил бы равенство. Одно несогласие со словом contract не является falsifier.

20.2. Falsifier readset claim

Предъявить implementation, которая не читает различающую coordinate, но для inputs, идентичных на всём её readset, выдаёт два различных deterministic outputs без скрытого state. Если найден clock, random seed, cache или environment, они входят в расширенный input.

20.3. Falsifier NSA-5

Предъявить finite positive nonnull instance, удовлетворяющий обоим hypotheses, least closure from bottom и member SCC, появившийся на конечной итерации. Nullary introduction, preseeded member, negative premise, nonleast fixed point или hidden base нарушают admission gates.

20.4. Falsifier label inertness

Предъявить contract, в котором mere substring occurrence является явно объявленным producer rule. Тогда label получает effect именно через это rule. Это positive control, а не автономная сила букв.

21. Failure classes

Следующие codes не являются 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.

Они выводят instance из точного admission path либо локализуют отсутствующую coordinate.

22. Самоприменение

Строка ADR-5 не допускает ADR-5. Строка NSA-5 не создаёт least closure. Этот файл является written record. Его формулы получают роли внутри объявленного reading contract. Чтобы checker принял formalization, нужны:

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

Даже успешный verdict будет иметь форму:

ACCEPTED - PROOF - UNDER - DECLARED - CONTRACT

Он не предъявит названное вне record. Самоприменение не разрушает теорему. Оно запрещает ей присвоить исключение.

23. Предел результата

Установлено:

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

Установлено:

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

Не установлено:

- что любой contract должен быть отвергнут;
- что formal derivations бесполезны;
- что compiler не может собирать artifact;
- что runtime не может выдавать trace;
- что axioms запрещены;
- что rules запрещены;
- что named external object отсутствует;
- что purpose отсутствует.

Сильный вывод точнее полного отрицания:

аксиома, правило, parser, checker, compiler, runtime и meta-proof получают admission только через записанные координаты; ни cycle, ни payload label не создаёт admission без declared base либо protected producer. Объявленная axiom-coordinate может служить base именно как input контракта, но не как самопорождающееся свойство линии.

24. Итоговый нож

Сначала записаны линии. Затем записано, как разрезать линии на tokens. Затем записано, как собрать tokens в formulas. Затем записано, какие formulas назвать axioms. Затем записано, какие переходы назвать rules. Затем записано, какой trace назвать proof. Затем записано, какой checker назвать accepted. Затем записано, какой meta-proof допускает checker. Цикл этих записей не создаёт первую запись допуска. Он только возвращает чтение к уже записанному.

Доказательство не находится внутри линий до правила чтения линий.

Правило чтения не находится внутри себя до правила его допуска.

Допуск остаётся координатой записи.

PREDECESSORS-AS-PREMISES: NONE.

04. ФОРМАЛЬНЫЙ ВЫВОД НЕ ПРЕДЪЯВЛЯЕТ РАКЕТУ

Binding, применение, цель, Марс и смысл через 100/200/300 лет

Phase 5 • техническое применение • 29 августа 2026

Кирилл Шебетов • beforeword / research

Формула может быть принята как вывод.

Это не предъявляет названное формулой.

Работающая ракета — также запись в данном предложении.

Аннотация

Этот файл проводит typed-разрез:

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

Стрелки в этой записи не выполняют переходы. Каждая стрелка требует отдельной protected coordinate. Reader-facing exposition label сохраняет scope:

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

Это reader-gloss, не runtime enum: executable checker записывает proof_status: ACCEPTED, а положительный terminal witness — kind: TARGET-ACCEPTED. Gloss не переименовывается в TRUE. Отказ checker также не переименовывается в FALSE. Application binding моделируется отдельным opaque code. Два bindings обязаны сохранять один derivation_core_sha256. Если при неизменном derivation core они дают разные application statuses, binding-coordinate входит в readset. Recorded binding не предъявляет названный объект. Он предъявляет ещё одну запись связи. Ракета, полёт, Марс, цель и смысл входят в формальную модель только как typed opaque codes. Их labels не получают исключения. Максимальный positive control допускает всю инженерную цепь:

- derivation принят;
- binding принят;

- artifact собран;
- runtime trace принят;
- trajectory criterion совпал;
- goal-code присутствует;
- endorsement-code присутствует.

Даже полный набор устанавливает только declared relations между records. Вопрос для чего? не выводится из отсутствия goal-code. Он также не выводится из наличия goal-code. Сто, двести и триста лет не являются constructors отсутствующего типа.

PREDECESSORS-AS-PREMISES: NONE.

0. Нулевая остановка на словах

Сначала предъявлены строки:

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

Слово истина не предъявляет истину. Слово применение не выполняет применение. Слово работает не запускает artifact. Слово ракета не предъявляет ракету. Слово летит не предъявляет полёт. Слово Марс не предъявляет планету. Слово цель не выбирает цель. Слово зачем не предъявляет ответ. Слово смысл не предъявляет смысл. Числа 100, 200 и 300 являются линиями в record. Их временная роль появляется только после horizon binding. Ни одна следующая уступка не отменяет эту остановку.

1. Typed carriers

Фиксируются попарно различные конечные 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 .

Одинаковые payload strings не объединяют sorts.

$$MARS \in F \neq MARS \in P.$$

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

Последняя правая часть здесь сама является supplied phrase. Carrier membership не выходит за carrier.

2. Ораque означает точную границу

Code называется ораque относительно checker, если checker вправе:

- проверить type tag;
- проверить canonical bytes;
- проверить equality;
- проверить digest;
- проверить declared relation membership;
- перенести code как payload;

но не вправе:

- извлечь из label названный объект;
- принять сходство имени за binding;
- создать цель из destination;
- создать endorsement из goal;

- создать смысл из возраста record.

Ораque не означает не существует. Это запрет незаписанной операции.

3. Proof core и его digest

Proof contract:

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

Для bytes $p \in B$ parser выдаёт derivation-code $d \in D$. Для target $f \in F$ вводится:

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

Его публичное чтение:

VALID-DERIVATION-IN-C

He:

TRUE

Здесь h_p обозначает ровно runtime field `derivation_core_sha256`, а не digest всей abstract tuple C_p .

Checker строит named canonical object, не tuple или array:

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

Пусть D_p обозначает этот exact named object. Тогда:

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

Canonicalizer задаёт порядок object keys; порядок key appearance в source JSON не является coordinate. Порядок members protected arrays остаётся частью value. rules здесь означает полный normalized rule catalog, а не только enabled subset. Digest сообщает code canonical runtime object under declared algorithm; он не является injective identity proof. Digest не предъявляет содержание label. Digest не выполняет parser. Digest не доказывает application.

4. Exact binding model

Следующая finite tuple exact внутри application extension этого файла; она не объявляется входной schema executable checker. Его frozen export_bindings input projection содержит fields:

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

Rich coordinates ниже либо проецируются в эти opaque codes, либо остаются отдельно записанными exposition records.

Binding record имеет type:

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

Здесь:

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

Binding acceptance:

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

Application contract:

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

Здесь:

- $J \subseteq J$ — admitted binding records;

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

Каждая coordinate является input. Строка β не выполняет binding сама.

Executable Phase 5 checker сохраняет более узкую public projection: `proof_status` принимает только ACCEPTED, REJECTED или UNPARSED; `binding export-status` принимает только EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE либо EXPLICIT-BINDING-RECORD-WITHOUT-ACCEPTED-TRACE. Эти runtime labels фиксируют co-occurrence records. Они не предъявляют bound referent. Остальные uppercase labels этого файла являются квалифицированными exposition-level diagnoses, а не обещанием новых runtime enum fields.

5. Recorded binding не предъявляет bound object

Предъявлен record:

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

Это устанавливает membership пары в таблице β . Оно не устанавливает:

$$f = o.$$

Оно не устанавливает:

$$o = \text{названное}.$$

Оно не устанавливает:

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

Даже field:

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

сначала является written field. Schema может принять его как opaque object-code. Schema acceptance не превращает field в объект вне schema.

6. BCR-5: теорема binding-coordinate readset

Пусть $j_0, j_1 \in J$ удовлетворяют:

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

Пусть все proof coordinates совпадают. Пусть bindings различаются:

$$\beta_0 \neq \beta_1.$$

Пусть application procedure F_A выдаёт:

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

Тогда:

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

И не существует функции:

$$g(h_p)$$

одновременно равной обоим различным outputs.

Доказательство

Если g читает только h_p , его input в двух запусках одинаков. Deterministic output обязан совпасть. Но outputs различны. Следовательно, proof core недостаточен. Различающий binding входит в readset. ▢ BCR-5 не говорит, какой binding правильный. Он устанавливает dependency.

7. Два exact counterbindings

Один proof core и один вычисленный 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
```

Exact proof bytes не изменились. Proof contract не изменился. Target formula-code не изменился. derivation_core_sha256 не изменился. Изменились opaque bindings. Поэтому разные application outputs не находятся автономно в proof.

8. Null binding

Допустим третий record:

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

Тогда status:

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

не противоречив. Valid derivation не требует application binding по одному своему type. Binding может быть добавлен позднее. Это revision record. Он не был скрыт внутри прежней формулы.

9. NOC-5: no export without constructor

Объявляется 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 имеют exact typed heads. Если:

1. initial set $X_0 \subseteq N_D$;
2. ни одно enabled rule не имеет head в $N_B \cup N_X \cup N_Z \cup N_L \cup N_G \cup N_E \cup N_S$;
3. closure строится positive least-fixed-point iteration;

то:

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

Доказательство

На шаге 0 outside-sort members отсутствуют. На каждом следующем шаге новый member появляется только как head enabled rule. Таких heads нет. Индукция сохраняет пустое пересечение.

▫ Это **NOC-5**. NOC-5 является supplementary source-free closure companion к separating readset result NEX-5 из файла 02, а не specialization или переопределением NEX-5. Теорема относится только к frozen tuple (N, X_0, R, E) , где E записана field enabled_rule_ids. Catalog constructors является declared completion boundary; NOC-5 не устанавливает отсутствие constructor или названного за этой boundary. Теорема не запрещает bridge. Она требует записать constructor. Если два bridge records при fixed core дают разные outputs, различающая bridge-coordinate необходима по BCR-5/NEX-5; constant или extensionally equivalent bridges сами по себе такую dependency не устанавливают.

10. Полная цепь условных статусов

Допустимы следующие predicates:

$$D = VD_{C_P}(p, f),$$

$$B = BIND_{C_A}(j, h_p, f),$$

$$X = BUILD_{C_A}(j, x),$$

$$Z = RUN_{C_A}(x, z),$$

$$L = MATCH_{C_A}(z, \ell),$$

$$G = GOAL_{C_A}(j, g),$$

$$E = ENDORSE_{C_A}(j, e),$$

$$S = MEANING_{C_A}(j, h, s).$$

Каждый predicate может быть принят. Но:

$$D \nRightarrow B,$$

$$B \nRightarrow X,$$

$$X \nRightarrow Z,$$

$$Z \nRightarrow L,$$

$$L \nRightarrow G,$$

$$G \nRightarrow E,$$

$$E \nRightarrow S.$$

Каждая стрелка становится допустимой только после exact bridge.

11. Четыре состояния derivation/application

Пусть:

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

Допустимы четыре pairs:

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

Pair	Ехакт чтение
$(0,0)$	derivation не принят; application chain не принят
$(1,0)$	derivation принят; binding или application chain отсутствует/отвергнут
$(0,1)$	application records приняты независимо от данного derivation
$(1,1)$	оба local statuses приняты

Числа — audit bits. 1 не означает TRUE. 0 не означает FALSE. Связь между bits требует policy.

12. Четыре состояния goal/endorsement

Пусть:

- $G=1$ — protected goal-code присутствует;
- $E=1$ — protected endorsement-code присутствует.

Допустимы:

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

$(0,0)$: ни один code не принят. $(1,0)$: цель записана, endorsement не записан. $(0,1)$: endorsement ссылается на иной goal-score либо instance invalid. $(1,1)$: оба code приняты. Наличие goal-code не является endorsement. Отсутствие endorsement-code не является objection. Отсутствие objection-code не является endorsement.

13. «Истина» как промежуточный payload label

Рассмотрим:

VALID-DERIVATION-IN-C
therefore TRUE

Вторая строка не является выводом первой без rule. Если contract вводит:

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

получен local label:

TRUE-LABEL-ASSIGNED-IN-C

Он остаётся scoped. Внешний безобластной truth-status в этой работе не вводится. Label assignment не предъявляет named referent.

14. Ракета как opaque code

Объявим:

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

Это exact code. Его canonical bytes можно хешировать. Его equality можно проверить. Его presence в binding можно проверить. Но:

$$o_R \in O \neq \text{названное предъявлено}.$$

Строка ракета есть не меняет type. Даже фраза:

ракета существует и летит

в этом файле является строкой. Её грамматическая уверенность не является bridge.

15. Полёт как 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

Он не сокращается до:

FLIGHT-OUTSIDE-RECORD-PRESENTED

Такого constructor в модели нет. Добавление его имени не добавит constructor.

16. Марс как destination code

Объявим:

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

Binding:

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

Trajectory criterion может ссылаться на m_R . Goal-code может ссылаться на m_R . Story payload может ссылаться на m_R . Ссылки остаются records.

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

Destination отвечает только на вопрос, записанный в своём contract. Он не отвечает автоматически на для чего?.

17. Максимальная инженерная уступка

Допустим одновременно:

1. $D = 1$;
2. binding schema valid;
3. core digest совпал;
4. compiler relation принят;
5. artifact identity принят;
6. runtime trace принят;
7. comparator вернул match;
8. trajectory-code принят;
9. destination-code равен MARS-CODE;
10. goal-code присутствует;
11. endorsement-code присутствует;
12. horizon-code присутствует.

Ни один status не отнимается. Итог:

FULL - DECLARED - CHAIN - ACCEPTED - IN - C

Этот итог сильнее ничего не произошло. Но он точнее фразы математика предъявила ракету. Он предъявляет accepted record chain.

18. Positive controls

18.1. Формальный control

При всех proof gates checker обязан принять exact derivation. Если он отвергает его, это implementation/test failure.

18.2. Binding control

При совпадшем h_p , schema-valid j и admitted β status binding обязан быть положительным.

18.3. Build control

Если $(j, x) \in \kappa$, build status принимается.

18.4. Runtime control

Если $(x, z) \in \rho$, runtime status принимается.

18.5. Match control

Если comparator relation выполнено, match status принимается.

18.6. Goal control

Если g находится в protected goal field и проходит scope gate, goal-code считается present. Positive controls запрещают риторическое стирание техники. Они не разрешают межтиповой экспорт.

19. Counterbindings

Counterbinding меняет только β . Proof core остаётся byte-identical. Допустим:

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

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

Пусть:

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

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

Если $z_A \neq z_B$, proof derivation не различил outputs. Их различил binding. Даже если $z_A = z_B$, equality trace-codes не удаляет binding coordinate. Она остаётся частью producer path.

20. Цель не выводится из траектории

Trajectory record может совпасть с criterion. Goal record может отсутствовать. Это даёт:

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

Вторая строка не равна:

```
NO - GOAL
```

Goal record может присутствовать. Тогда получен:

```
GOAL - CODE - PRESENT - IN - C
```

Он не равен:

```
GOAL - JUSTIFIED
```

Justification требует отдельного relation.

21. Вопрос для чего?

Вопрос входит как exact query-code:

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

Answer record:

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

Acceptance:

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

Наличие answer-code не предъясняет смысл. Оно устанавливает answer relation in contract. Отсутствие answer-code не предъясняет бессмысленность. Оно устанавливает только:

NO - ANSWER - CODE - IN - DECLARED - SCOPE

22. No negative from silence

Пусть:

$$g \notin G_C.$$

Это не вводит:

$$NO - GOAL.$$

Пусть:

$$e \notin E_C.$$

Это не вводит:

$$NOT - ENDORSED.$$

Пусть:

$$s \notin S_C.$$

Это не вводит:

$$MEANINGLESS.$$

Negative code требует own producer. Silence не является producer. Timeout не является silence certificate. Search failure не является exhaustive absence.

23. Horizon records 100/200/300

Объявлены:

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

Labels:

100-YEARS
200-YEARS
300-YEARS

не выполняют течение времени. Они получают horizon role через Ω . Пусть sequence records:

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

Переход:

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

Если T_C не имеет head constructor в meaning sort, то:

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

при условии:

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

Это induction по record steps. Это не утверждение о названных годах вне record.

24. HRZ-5: время не является type constructor

HRZ-5

Для positive typed transition system при base hypothesis $X_0 \cap N_S = \emptyset$ и при условии, что ни одно enabled rule не имеет head в N_S :

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

Доказательство

Base case задан пустым intersection. Induction step читает heads enabled rules. Ни один head не принадлежит N_S . Следовательно, новый meaning-code не появляется. □ При добавлении rule на шаге 201 meaning-code может появиться. Тогда его origin — новый rule/base record. Число 201 не создало rule.

25. Повторение не добавляет координату

Пусть один accepted technical record копируется n раз. В ledger notation:

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

Если canonical set semantics дедуплицирует copies:

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

Если ledger сохраняет multiplicity:

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

Ни equality, ни multiplicity не вводят goal/endorsement/meaning bridge. Повторение может быть premise явного threshold rule. Тогда threshold rule является input.

26. 100/200/300 лет и ракета

Допустим records сообщают:

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

Это три записи. Можно проверить их schema. Можно проверить их ordering. Можно проверить их digests. Можно вывести:

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

Из этого не следует:

```
MEANING-ACCUMULATED
```

Для последнего нужен constructor.

27. Destination, goal и meaning не образуют ladder

Часто записывается:

Марс $\Rightarrow$ цель $\Rightarrow$ смысл.

В Phase 5 это три different sorts. Допустимы состояния:

- destination есть, goal отсутствует;
- destination отсутствует, goal есть;
- goal есть, endorsement отсутствует;
- endorsement есть, meaning-answer отсутствует;
- meaning-answer есть без rocket binding;

- вся цепь присутствует.

Ни одно состояние не уничтожается словами очевидно зачем. Restriction возможно. Оно должно быть записано.

28. Compiler/checker layer как record

Checker artifact:

$$x_x \in X.$$

Compiler record:

$$c_x = (source_digest, compiler_digest, options, x_x).$$

Runtime record:

$$z_x = (x_x, input_digest, output_digest, environment_digest).$$

Этого достаточно для exact readset boundary. Нет необходимости бесконечно перечислять hardware layers. Если другая layer различает outputs, она добавляется как coordinate witness. Список не заменяет theorem. Theorem требует только различающую coordinate.

29. Falsifiers

29.1. Falsifier BCR-5

Предъявить одинаковый h_p , разные β_0, β_1 , разные deterministic outputs и функцию, которая читает только h_p , но воспроизводит оба outputs без hidden input.

29.2. Falsifier NOC-5

Предъявить positive least-closure instance без outside-sort initial member и без outside-sort rule head, в котором outside-sort member появился на конечной итерации.

29.3. Falsifier HRZ-5

Предъявить exact transition system, удовлетворяющий hypotheses, и первый t , на котором появился meaning-code без meaning-head rule или initial meaning-code.

29.4. Falsifier opaque-code boundary

Записать explicit constructor, превращающий object-code в новый protected status. Тогда status возникает по constructor. Это подтверждает необходимость binding. Строка но ракета же есть НЕ является schema witness. Она является новой строкой о ракете.

30. Limits

Файл не устанавливает:

- отсутствие названной ракеты;
- отсутствие названного полёта;
- отсутствие Марса;
- отсутствие целей;
- отсутствие endorsement;
- отсутствие смысла;
- невозможность инженерного применения;
- ложность формальной математики;
- происхождение математики;
- течение 100/200/300 лет.

Он устанавливает более точную границу:

ни один из этих outputs не извлекается из VALID-DERIVATION-IN-C без отдельной typed coordinate.

И ещё:

наличие recorded binding не предъявляет bound referent; оно предъявляет binding record.

31. Самоприменение

Этот файл содержит слова:

ракета
полёт
Марс
цель
смысл

Он не предъявляет названное ими. Формула BCR-5 не предъявляет binding вне файла. Формула NOC-5 не выполняет closure. Формула HRZ-5 не выполняет течение времени. Чтобы принять proofs, требуется declared proof contract. Чтобы применить outputs, требуется binding contract. Чтобы связать текст с named referent, требуется отдельная relation. Сам файл не получает исключения.

32. Итоговый нож

Сначала записана формула. Затем записано правило её вывода. Затем checker записал local acceptance. Затем binding записал имя ракета. Затем runtime записал trace. Затем comparator

записал совпало. Затем goal field записал Марс. Затем horizon field записал 300 лет. Затем вопрос записал зачем?. Каждый шаг добавил запись. Ни один шаг не вышел из записи своим названием.

VALID-DERIVATION не предъявляет истину.

Binding не предъявляет bound object.

Полёт не выбирает цель.

Триста лет не создают смысл.

PREDECESSORS-AS-PREMISES: NONE.

05. ВРАЖДЕБНАЯ ПРОВЕРКА И САМОПРИМЕНЕНИЕ PHASE 5

ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

0. Назначение и нулевая уступка метке

Этот файл пытается разрушить PSR-5, ADM-5 и NEX-5 в их точной области.

Письменная форма доказательство сначала входит как последовательность знаков.

Она не принимается за предъявление доказанного.

Письменная форма теорема сначала входит как последовательность знаков.

Она не принимается за установленный результат.

Письменная форма проверено сначала входит как последовательность знаков.

Она не принимается за выполненную проверку.

Строка `proof_status = ACCEPTED` является различимым `code-status`.

Она не вводит безусловный статус `PROOF`.

Она не вводит безусловный статус `TRUE`.

Она не вводит референт.

Она не вводит мировой факт.

Она не вводит применение.

Она не вводит цель применения.

Она не вводит смысл.

Главный объект красной команды — не слово о доказательстве, а точная зависимость результата от прочитанного входа.

1. Exact object Phase 5

Фиксируется конечный exact contract

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

Здесь Σ — конечный алфавит code units.

F — конечный sort формул-кодов.

$P = (\ell_0, \dots, \ell_{n-1})$ — конечная sequence supplied exact line-records.

$\delta_C(i, \ell_i)$ — объявленный deterministic partial per-line decoder. На reached index он либо выдаёт $\varphi_i \in F$, либо завершается UNPARSED.

$Ax_C \subseteq F$ — exact множество axiom-codes.

R_C — exact конечный каталог positive ground set-premise rules.

$E_C \subseteq \{id(r) : r \in R_C\}$ — exact множество enabled_rule_ids; только rules с ids из E_C участвуют в admission.

$g_C \in F$ — exact goal-code.

$mode_C$ фиксирует declared режим чтения, goal matching и допустимые syntactic conventions.

Это математическая projection, а не список literal runtime key names и не заявление byte-identity с JSON object.

Field map к runtime:

- P соответствует protected surface_record = {surface_id, lines};
- F соответствует ids из statement_codes;
- δ_C соответствует exact decoding, где каждый item повторяет line_index, line_text и даёт statement_code;
- Ax_C соответствует axiom_codes;
- R_C соответствует полному rules catalog;
- E_C соответствует enabled_rule_ids;
- g_C соответствует target_code;
- $mode_C$ соответствует frozen top-level contract policies.

Runtime key payload в protected admission projection не входит.

Каждая reached decoded строка допускается только одним из двух путей:

$$LineOK_C(P, i)$$

если $\varphi_i = \delta_C(i, \ell_i) \in Ax_C$, либо если существует exact ground rule $r \in R_C$ с $id(r) \in E_C$, чьи premise codes имеют accepted witnesses только на индексы $j < i$ и дают head φ_i .

Финальный допуск:

$$Accept_C(P)$$

имеет место ровно тогда, когда traversal достигает конца, decoder определён на каждой supplied line, все строки имеют допустимые witnesses, а финальная строка совпадает с g_C в объявленном goal-mode.

Exact failure precedence также sequential: на каждом reached index сначала missing decode даёт UNPARSED; mapped, но inadmissible line даёт REJECTED; любой terminal event останавливает run, поэтому later defect не переписывает earlier status. Только после полного traversal проверяется target.

Никакая строка payload не подменяет $\Sigma, F, \delta_C, Ax_C, R_C, E_C, g_C, mode_C$.

2. Admission gates

Атака считается атакой на Phase 5 только после прохождения gates.

2.1. FINITE-EXACT-SURFACE

Точные code units P предъявлены.

Их порядок предъявлен.

Их границы предъявлены.

Нормализация либо запрещена, либо включена в contract.

2.2. FINITE-TYPED-CONTRACT

Σ, F , axioms, полный rule catalog, enabled_rule_ids, arities, goal и mode конечны и типизированы.

Формула-код не становится правилом из-за имени.

Правило-код не становится применённым правилом из-за payload.

2.3. DECLARED-DETERMINISTIC-PARSER

Parser и version parser предъявлены как coordinate contract.

Если parsing допускает несколько деревьев, tie-break является частью mode.

Скрытый выбор parser не проходит gate.

2.4. PRIOR-CITATIONS-ONLY

Rule witness строки i может ссылаться только на индексы $j < i$.

Циклическая ссылка на собственный вывод не проходит gate.

Ссылка на будущую строку не проходит gate.

2.5. EXACT-AXIOMS

Membership в Ax_C пересчитывается.

Поле axiom: true внутри payload не создаёт membership.

2.6. EXACT-RULES

Rule name не заменяет exact premise_codes, conclusion_code и membership rule id в enabled_rule_ids.

Каждый ground rule пересчитывается по exact catalog record.

Наличие rule в полном catalog не делает его enabled: id должен входить в E_C .

2.7. EXACT-GOAL

Goal-code предъявлен отдельно от proof surface.

Поле goal_reached: true не заменяет финальное сопоставление.

2.8. DISTINCT-STATUS-CODES

Parser failure, contract failure, rejected derivation и accepted derivation не склеиваются.

proof_status = ACCEPTED не сокращается до TRUE.

2.9. FROZEN-READSET

(P, C) фиксируется до просмотра результата, включая полный R_C и enabled subset E_C .

Post hoc добавление axioms, rules или goal означает новый contract.

2.10. CERTIFICATE-RECOMPUTATION

Parse digest, line witnesses и goal witness пересчитываются.

Заявленный certificate не принимается по собственному имени.

2.11. NEX-TYPED-TARGET

Для NEX-5 внешний target задаётся отдельным typed status-space.

Слова референт, истина, применение, цель и смысл не заменяют этот target-space.

2.12. NEX-SEPARATING-COMPLETIONS

Две external completions сохраняют одинаковые (P, C) , но дают разные target-status.

Различие должно быть предъявлено в отдельной bridge/application coordinate.

2.13. NO-PAYLOAD-AUTHORITY

Self-label не изменяет protected fields.

Хеш payload не превращает payload в axiom или rule.

2.14. BOUNDED-CLAIM

Выход относится к exact record и exact contract.

Он не распространяется молча на все записи, все parser-ы или мир.

Нарушение gate выводит пример из theorem-domain.

Оно не является falsifier теоремы.

3. Три результата, которые требуется атаковать

3.1. PSR-5 — proof-status readset theorem

Пусть один и тот же exact surface P имеет разные exact значения canonical ternary status function $V_C(P) \in \{ACCEPTED, REJECTED, UNPARSED\}$:

$$V_{C_0}(P) \neq V_{C_1}(P).$$

Тогда selector, читающий только P , не может вернуть оба результата для ordered inputs (P, C_0) и (P, C_1) .

Следовательно, различающая contract-coordinate входит в полный readset статуса.

PSR-5 не говорит, что каждый pair contracts обязан различать status.

PSR-5 не говорит, что bytes не важны.

PSR-5 говорит, что bytes недостаточны там, где одинаковые bytes допускают разделяющую пару contracts.

3.2. ADM-5 — finite admission certificate

Пусть $W_C(P) = (T, \tau)$ — generated output contract evaluation: finite trace T и terminal_witness τ . Runtime не получает $W_C(P)$ как input.

При gates Phase 5:

$$Accept_C(P) \Leftrightarrow CertOK_C(P, W_C(P)) \wedge \tau.kind = TARGET - ACCEPTED.$$

Reference certificate объединяет derivation_core_sha256, generated trace и final terminal witness.

Certificate конечен.

Certificate проверяем относительно (P, C) .

Certificate не является внешней семантической аттестацией.

3.3. NEX-5 — no external export without bridge

Пусть $ADM_C(P) = 1$ и runtime proof_status = ACCEPTED. External completions K_0, K_1 сохраняют этот один exact accepted (P, C) , но export target-status function X различается:

$$X(P, C, K_0) \neq X(P, C, K_1).$$

Selector, читающий только (P, C) , не может вернуть оба target-status.

Значит, separating bridge/application coordinate входит в полный readset target-status.

Formal admission сам по себе не предъявляет эту coordinate.

4. Status discipline красной команды

Для каждой атаки фиксируются четыре строки.

RECORD-STATUS — точный статус записи.

ESTABLISHED — что вычислено внутри объявленного объекта.

NOT-INTRODUCED — что не появилось от этого вычисления.

FALSIFIER-TEST — какой результат действительно разрушил бы theorem.

Строки status, proof_status, terminal_witness.kind, export_status, comparison_status и четыре fences ниже воспроизводят exact vocabulary checker.

Uppercase labels для compiler, signature, title, test, history и других расширенных objection contracts не являются полями proof_readset_checker.py; это явно scoped record-classifications соответствующей атаки.

Они не подменяют runtime vocabulary.

Ниже нет неоговорённого PROOF.

Ниже нет неоговорённого TRUE.

Ниже нет неоговорённого FALSE о мире.

5. Атака А — одни bytes, два parser-contracts

Surface:

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

Contract C_0 содержит total exact decoding трёх indexed lines в codes $s:A$, $s:I_AB$, $s:B$.

Contract C_1 сохраняет те же surface-lines, но его partial decoding не содержит mapping последней line.

В C_0 первые две строки — axioms, а modus ponens — declared rule.

В C_1 checker останавливается на первом unmapped surface index.

Bytes одинаковы.

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = UNPARSED; terminal_witness.kind = UNMAPPED-SURFACE-LINE.

ESTABLISHED: exact decoding-coordinate различает admission outputs.

NOT-INTRODUCED: ни один output не вводит референт a , референт b или truth-value вне contract.

FALSIFIER-TEST: bytes-only selector должен без чтения contract вернуть два различных ordered outputs для одного exact P .

Такой selector должен удовлетворить одновременно $z(P)=0$ и $z(P)=1$.

Предъявление только одной ветви не falsifies PSR-5.

6. Атака В — tokenization variant

Surface:

```
p->q
```

Parser C_0 читает p , $->$, q .

Parser C_1 читает один atom $p->q$.

Goal C_0 — implication-code.

Goal C_1 — atom-code.

Даже если обе ветви accepted, их parsed objects различны.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = ACCEPTED.

ESTABLISHED: два contracts могут присвоить одному surface два разных typed parse.

NOT-INTRODUCED: СОВМЕСТНОЕ СЛОВО accepted не устанавливает тождество parsed formula-codes.

FALSIFIER-TEST: PSR-5 здесь не активирован, если exact acceptance-bit совпал.

Если claimed output включает parse identity, различие снова требует parser-coordinate.

Совпадение одного status-bit не удаляет contract из полного certificate readset.

7. Атака C — whitespace и normalization

Surface содержит exact sequence с двумя пробелами.

C_0 игнорирует repeated whitespace.

C_1 считает второй пробел malformed token.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = UNPARSED.

ESTABLISHED: normalization-policy находится в readset.

NOT-INTRODUCED: слово пробел не предъявляет физическое пустое место; здесь учитывается code unit.

FALSIFIER-TEST: одинаковый surface-only output должен совпасть с двумя различными contract-relative results.

Невидимая pre-normalization не является исключением.

Она является скрытым contract.

8. Атака D — одни bytes, разные axioms

Surface:

а

В C_0 : $a \in Ax_{C_0}$ и $g_{C_0} = a$.

В C_1 : $a \notin Ax_{C_1}$, rules empty, $g_{C_1} = a$.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = REJECTED.

ESTABLISHED: axiom membership является contract-coordinate.

NOT-INTRODUCED: axiom membership не предъявляет названное *a* и не устанавливает world truth.

FALSIFIER-TEST: функция только от bytes должна различить contracts, не читая различие axioms.

Поле внутри surface АКСИОМ *a* не помогает.

Оно сначала является ещё одной строкой surface.

Только parser и axiom policy дают ей роль.

9. Атака Е — одни bytes, разные rules

Surface:

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

C_0 содержит modus ponens.

C_1 не содержит modus ponens.

Axioms и goal одинаковы.

RECORD-STATUS(C0): proof_status = ACCEPTED.

RECORD-STATUS(C1): proof_status = REJECTED.

ESTABLISHED: rule-set различает admission.

NOT-INTRODUCED: применение modus ponens внутри code-system не связывает *a* или *b* с объектами.

FALSIFIER-TEST: rule-blind selector должен выдать два разных correct outputs на одинаковом readset.

Название modus ponens не является rule instance.

Нужны exact premise_codes, earlier accepted witnesses и conclusion_code.

10. Атака F — extension с другим side condition

Эта атака не является result исполняемого base kernel Phase 5. Base kernel допускает только positive ground set-premise rules без substitution и side-condition language.

Если расширение вводит side condition, оно создаёт новую protected coordinate contract. Тогда два extended contracts могут содержать rule с одинаковым именем *R*.

В C_0 side condition проходит.

В C_1 требуется дополнительный freshness-condition.

Surface одинаков.

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = REJECTED.

ESTABLISHED-IN-EXTENSION-ONLY: имя rule не исчерпывает exact extended schema.

NOT-INTRODUCED: равенство имён не устанавливает равенство правил.

FALSIFIER-TEST-FOR-EXTENSION: новая side-condition coordinate должна оказаться ненужной при разделённых outputs.

Если extended checker читает side condition, он читает extended contract.

Если base checker его не читает, это не defect base runtime: side-condition field не входит в его declared input language.

11. Атака G — один parse, разные goals

Surface и derivation graph одинаковы.

В C_0 goal равен финальной формуле b .

В C_1 goal равен c .

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = REJECTED; terminal_witness.kind = TARGET-LINE-MISMATCH.

ESTABLISHED: goal-coordinate входит в admission readset.

NOT-INTRODUCED: достижение goal-code не вводит цель действия или назначение объекта.

FALSIFIER-TEST: goal-blind selector должен корректно разделить outputs.

Слово цель имеет здесь только declared record-role.

Оно не предъявляет для чего.

12. Атака H — accepted derivation без binding

Contract допускает proof surface.

External binding-coordinate отсутствует.

Нет map от atom-codes к объектам.

Нет truth-assignment.

Нет application target.

Нет purpose-coordinate.

RECORD-STATUS: proof_status = ACCEPTED.

ESTABLISHED: generated $W_C(P)$ проходит CertOK, а terminal kind равен TARGET-ACCEPTED.

NOT-INTRODUCED: референт, мировой truth-status, применение, назначение и смысл.

FALSIFIER-TEST(NEX-5): из одного (P, C) должен вычисляться unique external target-status без любой bridge-coordinate, причём для всех separating completions.

Простое повторение accepted-status не выполняет этот test.

Оно остаётся на стороне (P, C) .

13. Атака I — один accepted proof, два bindings

Фиксируются одинаковые (P, C) .

Completion K_0 связывает atom r с target-id u .

Completion K_1 связывает atom r с target-id v .

$u \neq v$ в declared target sort.

Formal admission одинаков в обеих ветвях.

External target-status различен.

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: bridge-coordinate разделяет target outputs.

NOT-INTRODUCED: surface не выбирает между u и v .

FALSIFIER-TEST: selector только от (P, C) должен вернуть оба различающихся target-status.

Если binding записан внутри нового contract, input уже изменён.

Это подтверждает, а не снимает readset boundary.

14. Атака J — два truth assignments

Один formal derivation accepted.

Typed completion K_0 присваивает target-code значение t_0 .

Typed completion K_1 присваивает target-code значение t_1 .

Значения различны как code-members 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 сохраняется при различающихся external completions.

NOT-INTRODUCED: acceptance не выбирает assignment.

FALSIFIER-TEST: assignment-blind selector должен быть extensionally correct на обеих ветвях.

Слово истина в названии assignment не удостоверяет мир.

В атаке различаются только declared target-codes.

15. Атака K — inconsistent axioms и explosion

Contract содержит axiom-codes p и $\neg p$.

Contract содержит правило explosion.

Surface выводит arbitrary goal-code q .

Все line witnesses корректны относительно этого contract.

RECORD-STATUS: proof_status = ACCEPTED.

ESTABLISHED: q допускается в exact inconsistent calculus.

NOT-INTRODUCED: непротиворечивость, target binding, world truth q , цель или смысл.

FALSIFIER-TEST(ADM-5): proof_status = ACCEPTED, но generated $W_C(P)$ не проходит CertOK либо не заканчивается TARGET-ACCEPTED.

Explosion не falsifies ADM-5.

Он даёт generated trace внутри объявленных rules.

FALSIFIER-TEST(NEX-5): formal acceptance должен самостоятельно выбрать внешний target-status во всех separating completions.

Слово всё в неформальном описании explosion не расширяет target-space.

16. Атака L — paraconsistent variant

Тот же surface и те же axiom-codes подаются C_0 и C_1 .

C_0 содержит explosion.

C_1 — paraconsistent rule-set без explosion.

RECORD-STATUS(C_0): proof_status = ACCEPTED.

RECORD-STATUS(C_1): proof_status = REJECTED.

ESTABLISHED: rule-coordinate снова разделяет outputs.

NOT-INTRODUCED: слово логика не создаёт единственный обязательный rule-set.

FALSIFIER-TEST: PSR-5 разрушен только selector-ом, который не читает различие rule-sets и всё же выдаёт обе ветви.

Отсылка к привычной логике является выбором contract, а не свойством bytes.

17. Атака M — circular derivation

Surface утверждает, что строка 0 следует из строки 0.

Payload содержит label valid proof.

Claimed witness нарушает PRIOR-CITATIONS-ONLY и не входит в protected runtime input.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; recomputed trace либо содержит только earlier premise_witnesses, либо contract получает proof_status = REJECTED.

ESTABLISHED: record не находится в domain ADM-5 accepted branch.

NOT-INTRODUCED: ложность названного, истинность противоположного или внешний semantic status.

FALSIFIER-TEST: claimed circular witness не falsifies equivalence; нужен mismatch между proof_status и recomputed generated $W_C(P)$.

Self-reference label не устраняет индексный порядок.

18. Атака N — future-line citation

Строка 0 ссылается на строку 2.

Строка 2 ссылается на строку 0.

Parser успешно строит payload graph.

Но graph не удовлетворяет admission mode.

RECORD-STATUS: proof_status = REJECTED.

ESTABLISHED: parse success не равен line admission.

NOT-INTRODUCED: rejection не вводит negative formula-code.

FALSIFIER-TEST: generated trace содержит premise witness с $j \geq i$, но CertOK или runtime всё же принимает его при strict-prefix policy.

Если другой contract допускает cyclic proofs, это другой mode.

PSR-5 применим к разделяющей паре modes.

19. Атака O — self-label spoof

Input payload содержит:

```
status: proof_status = ACCEPTED
theorem: PROVED
world_truth: true
```

Protected parser, axioms, rules и goal не берут эти поля как authority.

Если spoof добавлен только в payload: RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS, а proof_status пересчитывается из protected fields. Если spoof ломает protected shape: RECORD-STATUS: INVALID-PROOF-READSET-RECORD.

ESTABLISHED: self-label не изменяет recomputed result.

NOT-INTRODUCED: proof status, theorem result или world truth из payload.

FALSIFIER-TEST: spoof должен изменить recomputed protected output при неизменном protected readset.

Если schema разрешает payload, он сохраняется только как uninterpreted field.

Если schema запрещает payload, record отклоняется.

Обе ветви сохраняют boundary.

20. Атака P — certificate spoof

Certificate перечисляет неверный parse digest.

Line witnesses выглядят синтаксически правдоподобно.

Goal witness ссылается на отсутствующую строку.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; claimed certificate bytes inert, а exact trace и terminal_witness пересчитаны.

ESTABLISHED: claimed certificate bytes не входят в protected readset; generated $W_C(P)$ пересчитан.

NOT-INTRODUCED: внешний proof authority или semantic status.

FALSIFIER-TEST: ADM-5 falsified только при расхождении exact proof_status и CertOK generated $W_C(P)$ с terminal condition.

Один claimed certificate не меняет generated output.

Recomputed generated trace закрывает только admission относительно (P, C) .

21. Атака Q — checker как доказательная машина

Возражение: checker возвращает accepted, значит доказанное предъявлено.

Checker получает serialised (P, C) .

Checker реализует parser и admission algorithm.

Он возвращает новый serialised record.

RECORD-STATUS: CHECKER-OUTPUT-MATCHES-DECLARED-ADMISSION-ALGORITHM.

ESTABLISHED: воспроизводимое отношение input-output внутри implementation boundary.

NOT-INTRODUCED: названный объект, world truth, безошибочность implementation, цель или смысл.

FALSIFIER-TEST(ADM-5): recomputed CertOK generated output и accepted predicate должны расходиться на admitted input.

FALSIFIER-TEST(NEX-5): checker без bridge должен выдавать правильные разные external target-status для separating completions, которых он не читает.

Надпись на кнопке Verify не расширяет readset.

22. Атака R — compiler objection

Возражение: компилятор превращает символы в работающий machine code.

Compiler читает source относительно grammar, type rules, target ISA и build options.

Изменение options может изменить output при одинаковом source.

Изменение ISA может сделать output invalid.

RECORD-STATUS: COMPILED-UNDER-DECLARED-TOOLCHAIN-CONTRACT.

ESTABLISHED: output bytes получены данным toolchain pipeline.

NOT-INTRODUCED: referent source-имён, world truth комментариев, назначение программы или смысл запуска.

FALSIFIER-TEST(PSR-5): source-only selector должен воспроизвести разные toolchain-relative outputs без чтения toolchain-coordinate.

FALSIFIER-TEST(NEX-5): compiled bytes должны выбрать external application target без deployment bridge.

Компиляция является поздней операцией над записью.

Она не делает source тем, что source называет.

23. Атака S — proof assistant objection

Возражение: kernel proof assistant проверяет theorem, поэтому contract больше не нужен.

Kernel сам задаёт parser, term language, typing rules, reduction и trusted primitives.

Version kernel и flags входят в operational contract.

RECORD-STATUS: KERNEL-ACCEPTED-UNDER-DECLARED-VERSION-AND-ENVIRONMENT.

ESTABLISHED: term имеет допустимый type-code в данном environment.

NOT-INTRODUCED: внешний референт type-name, world truth, физическое применение или смысл.

FALSIFIER-TEST: один term должен получить разделённые version-relative outputs без чтения version/environment.

Фраза small trusted kernel описывает размер TCB.

Она не удаляет TCB.

24. Атака T — soundness theorem

Возражение: calculus имеет доказанную soundness, значит accepted formula истинна.

Soundness proof — ещё один proof surface под своим contract.

Ero conclusion связывает syntactic derivability с model-relative satisfaction при exact definitions.

RECORD-STATUS: SOUNDNESS-DERIVATION-ACCEPTED-UNDER-META-CONTRACT.

ESTABLISHED: meta-contract допускает записанное implication между code-predicates.

NOT-INTRODUCED: единственная внешняя model-binding, world truth или референт исходных atoms.

FALSIFIER-TEST(NEX-5): meta-proof без bridge должен различить target-status в separating external completions.

Перенос contract на meta-level не устраняет coordinate.

Он добавляет новый record и новый contract.

25. Атака U — categorical semantics

Возражение: единственность модели с точностью до изоморфизма фиксирует названное.

Фраза с точностью до изоморфизма уже ограничивает тип различимости.

Signature, axioms, morphisms и equivalence notion входят в contract.

RECORD-STATUS: MODEL-CLASS-CONSTRAINED-UNDER-DECLARED-SEMANTIC-CONTRACT.

ESTABLISHED: declared structures удовлетворяют recorded conditions в заданном semantics.

NOT-INTRODUCED: один внешний объект как названное, source-origin, цель или смысл.

FALSIFIER-TEST: одинаковый formal record должен выбрать bridge к одному external target при наличии separating bridges.

Изоморфизм — записанное отношение внутри модели.

Он не предъявляет объект за записью.

26. Атака V — mathematical effectiveness

Возражение: расчёт предсказывает измерение, поэтому линии несут истину.

Для сравнения нужны measurement protocol, units, tolerance, alignment и matching rule.

Эти координаты не содержатся автоматически в proof surface.

RECORD-STATUS: PREDICTION-MATCH-UNDER-DECLARED-MEASUREMENT-AND-TOLERANCE-CONTRACT.

ESTABLISHED: два record values попали в declared comparison relation.

NOT-INTRODUCED: тождество записей с названным, ultimate purpose или смысл.

FALSIFIER-TEST(NEX-5): proof surface без measurement bridge должен выбирать разные match-status в separating protocols.

Совпадение записанных чисел является предъявленным совпадением.

Название измеряемого не появляется из числового равенства.

27. Атака W — физическая ракета

Возражение: числа позволили построить ракету, ракета летит, значит математика предъявила доказанное.

Здесь смешаны по меньшей мере пять records:

1. formal proof или calculation surface;
2. engineering interpretation;
3. manufacturing instructions;
4. observation protocol;
5. purpose attribution.

Каждый переход имеет отдельную bridge-coordinate.

RECORD-STATUS: proof_status = ACCEPTED; world_export_status = NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER.

Application-link и observation-match являются отдельными scoped records вне checker.

ESTABLISHED: при предъявленных contracts можно связать records последовательностью declared relations.

NOT-INTRODUCED: назначение для чего, смысл полёта, тождество formula с ракетой или world authority числа.

FALSIFIER-TEST(NEX-5): один (P, C) должен определить purpose-status для двух completions с одинаковой ракетной траекторией и разными declared purposes.

Физическое различие после запуска не отменяет запись bridge.

Оно не превращает линии calculation в летящий объект.

Ответ чтобы долететь является новой строкой purpose-description.

Даже наблюдаемый полёт не содержит внутри себя строку для чего.

28. Атака X — одинаковая ракета, две цели

Фиксируется один technical record и одна наблюдаемая trajectory-record.

Completion K_0 содержит declared purpose-code RESEARCH.

Completion K_1 содержит declared purpose-code WEAPON.

Formal proof и trajectory-record одинаковы.

RECORD-STATUS(K_0): export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE; goal_code содержит один opaque id.

RECORD-STATUS(K_1): export_status = EXPLICIT-BINDING-RECORD-COOCURS-WITH-ACCEPTED-TRACE; goal_code содержит другой opaque id.

Pairwise status: BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS.

ESTABLISHED: purpose-coordinate разделяет outputs.

NOT-INTRODUCED: trajectory не выбирает purpose-code.

FALSIFIER-TEST: purpose-blind selector должен вернуть оба разных statuses на одном (P, C) и одной trajectory-record.

Слово исследование не предъявляет цель вне completion.

Слово оружие также не предъявляет её.

Атака использует только typed separation.

29. Атака Y — historical origin of mathematics

Возражение: математика возникла до языка, независимо от языка или из невербального количества.

Эти предложения входят как записи о происхождении.

Phase 5 не использует противоположное историческое предложение как premise.

PSR-5 относится к текущей dependence exact status от exact contract.

ADM-5 относится к текущему finite generated trace и его certOK relation.

NEX-5 относится к текущему missing bridge при separating completions.

RECORD-STATUS: `historical_origin_status` = NO-HISTORICAL-ORIGIN-ATTESTATION.

ESTABLISHED: историческая ветвь не входит в `premises` трёх результатов.

NOT-INTRODUCED: подтверждение или опровержение происхождения математики.

FALSIFIER-TEST: `historical claim falsifies Phase 5` только если из него следует `countermodel` точной `theorem`, проходящий её `gates`.

Слова `младенец`, `животное`, `количество` и `невербальное` не предъявляют названное.

Ссылка на `experiment` остаётся `citation-record`.

Она не заменяет `exact countermodel PSR-5`, `ADM-5` или `NEX-5`.

30. Атака Z — «количество существует без записи»

Предложение предъявляет `written form` `количество существует без записи`.

Phase 5 не выводит его отрицание.

Phase 5 также не принимает его за установленный `referent`.

RECORD-STATUS: `world_export_status` = NO-WORLD-EXPORT-CONSTRUCTOR-IN-DECLARED-CHECKER.

ESTABLISHED: `theorem-domain` ограничен `record-status` и `readset`.

NOT-INTRODUCED: существование или отсутствие количества вне записи.

FALSIFIER-TEST: нужен `counterexample` точной `readset theorem`, а не повторение `ontology sentence`.

Даже если `external item` будет отдельно предъявлен, слово `количество` не станет этим `item`.

31. Атака AA — human intuition

Возражение: математик видит истинность без формального `contract`.

Строка видит истинность сначала является `description-record`.

Если `intuition` даёт `admission output`, условия различения её `outputs` составляют `operational coordinate`.

RECORD-STATUS: INTUITION-ATTRIBUTION-NOT-A-FORMAL-CERTIFICATE.

ESTABLISHED: `attribution` не заменяет `generated  $W_C(P)$` и его `ADM checks`.

NOT-INTRODUCED: доступ к `named truth` или `authority субъекта`.

FALSIFIER-TEST: предъявить mismatch proof_status и generated $W_C(P)$ в domain ADM-5, а не изменить domain на intuition-mode.

Если mode разрешает oracle step, oracle входит в contract.

32. Атака AB — universal agreement

Возражение: все согласны с доказательством.

Список согласий является новым record family.

Квантор все требует declared finite или otherwise specified domain.

RECORD-STATUS: AGREEMENT-RECORD-SEPARATE-FROM-ADMISSION-RECORD.

ESTABLISHED: при точном roster можно вычислить agreement relation.

NOT-INTRODUCED: referent, world truth, source или смысл proof surface.

FALSIFIER-TEST: consensus должен устранить separating contracts или bridges без добавления readset-coordinate.

Добавление голосов не изменяет bytes исходного proof.

33. Атака AC — successful repetition

Checker запущен миллион раз.

Output каждый раз одинаков.

RECORD-STATUS: DETERMINISTIC-REPETITION-UNDER-FIXED-IMPLEMENTATION.

ESTABLISHED: повторяемость implementation relation на tested environment.

NOT-INTRODUCED: external referent, world truth, complete absence of bugs или смысл.

FALSIFIER-TEST: repetition должна позволить selector-у различить нечитанные contract или bridge coordinates.

Количество повторов не расширяет input readset.

34. Атака AD — tests passed

Test suite возвращает PASS.

Test cases и expected outputs являются records.

Runner сравнивает actual и expected по declared equality.

RECORD-STATUS: TEST-SUITE-PASS-UNDER-DECLARED-RUNNER-AND-CASES.

ESTABLISHED: tested input-output pairs совпали с recorded expectations.

NOT-INTRODUCED: truth названного, полнота test space, отсутствие всех defects или внешний authority.

FALSIFIER-TEST: найдён admitted counterexample theorem, а не только untested implementation branch.

Failing test может выявить implementation defect.

Passing tests не становятся theorem premise.

35. Атака AE — hash equality

Опубликован SHA-256 digest.

Повторное вычисление даёт тот же digest.

RECORD-STATUS: BYTES-MATCH-RECORDED-DIGEST-UNDER-DECLARED-HASH-ALGORITHM.

ESTABLISHED: byte sequence совпадает с hash-addressed release record в пределах выбранного algorithm relation.

NOT-INTRODUCED: истинность текста, авторство, происхождение, полнота, смысл или referent.

FALSIFIER-TEST: hash equality должна сама обеспечить external target-status при separating completions.

Hash фиксирует различимость bytes.

Он не делает bytes доказанным.

36. Атака AF — digital signature

Signature проходит cryptographic verification.

RECORD-STATUS: SIGNATURE-VALID-UNDER-DECLARED-KEY-AND-ALGORITHM.

ESTABLISHED: signature relation проходит verification для предъявленных bytes и key record.

NOT-INTRODUCED: truth payload, identity вне key-binding contract, purpose или meaning.

FALSIFIER-TEST: signature должна заменить contract и bridge, не вводя собственные coordinates.

Key binding уже является дополнительным record.

37. Атака AG — title authority

Публичный title:

ДОКАЗАТЕЛЬСТВО НЕ ПРЕДЪЯВЛЯЕТ ДОКАЗАННОЕ

Title является предъявленной строкой.

Он не является premise PSR-5.

Он не является certificate ADM-5.

Он не является bridge NEX-5.

RECORD-STATUS: UNBOUNDED-STANDALONE.

ESTABLISHED: существует exact heading occurrence в release bytes.

NOT-INTRODUCED: безусловная истинность heading, мировой закон или authority автора.

FALSIFIER-TEST: heading должен быть заменён countermodel exact theorem, а не встречным heading.

Сам title подчинён собственной границе.

Он не предъявляет доказательство.

Он не предъявляет доказанное.

38. Атака АН — theorem-label self-application

Labels PSR-5, ADM-5 и NEX-5 являются code-names.

Их presence не создаёт theorem status.

Theorem status появляется только после чтения definitions, premises и derivation under a contract.

RECORD-STATUS: THEOREM-LABELS-PRESENT-IN-RECORD.

ESTABLISHED: labels адресуют exact claim blocks.

NOT-INTRODUCED: truth, novelty, priority или external validity этих claims.

FALSIFIER-TEST: требуется exact admitted countermodel соответствующей claim, проходящий gates.

Phase 5 не освобождает собственные формулы от contract dependence.

Это не дефект самоприменения.

Это заявленная boundary.

39. Атака AI — checker-name self-application

Имя файла proof_readset_checker.py не предъявляет checker behavior.

Executable bytes и run environment являются отдельными inputs.

RECORD-STATUS: FILE-NAME-PRESENT до исполнения.

RECORD-STATUS: CHECKER-RUN-RECORDED только после exact command и output record.

ESTABLISHED: при воспроизведённом run вычислена contract-relative строка.

NOT-INTRODUCED: безошибочность, proof authority или world truth.

FALSIFIER-TEST: implementation output должен расходиться с reference predicate на admitted test; это implementation falsifier, не автоматически theorem falsifier.

Чтобы falsify theorem, расхождение должно образовать точный countermodel theorem statement.

40. Атака AJ — test-name self-application

Имя test_phase5_self_audit.py сначала является filename record.

Префикс test_ не исполняет тест.

Строка PASS в source не является run result.

RECORD-STATUS: TEST-SOURCE-PRESENT до запуска.

RECORD-STATUS: TEST-RUN-PASS-OR-FAIL после exact run.

ESTABLISHED: runner result относится к exact environment и cases.

NOT-INTRODUCED: полнота тестов или semantic truth claims.

FALSIFIER-TEST: конкретный failing invariant должен быть отделён от theorem counterexample.

41. Атака AK — manifest self-application

Manifest перечисляет files, sizes и digests.

Его собственное имя не доказывает completeness package.

Package-filelist и manifest policy должны быть предъявлены отдельно.

RECORD-STATUS: MANIFEST-CONSISTENT-WITH-DECLARED-PACKAGE-POLICY либо structured failure.

ESTABLISHED: перечисленные bytes совпадают с recorded digests.

NOT-INTRODUCED: отсутствие неучтённых файлов вне declared package boundary, truth рукописи или priority.

FALSIFIER-TEST: manifest не является premise PSR-5, ADM-5 или NEX-5; ero defect falsifies release-integrity claim в точном объёме.

42. Атака AL — self-hash

Файл не может наивно содержать обычный digest собственных финальных bytes без отдельной fixed-point scheme.

Phase 5 не требует такого объекта.

Manifest исключает собственную строку либо использует declared packaging rule.

RECORD-STATUS: HASH-SCOPE-DECLARED.

ESTABLISHED: digest относится к exact listed byte scope.

NOT-INTRODUCED: self-certifying truth.

FALSIFIER-TEST: scope mismatch falsifies hash claim, но не readset theorems.

43. Атака AM — self-audit claim registry

Self-audit читает exact occurrences claims.

Он проверяет hashes, required strings, forbidden strings и declared dependencies.

RECORD-STATUS: SELF-AUDIT-PASS-UNDER-DECLARED-REGISTRY.

ESTABLISHED: registered syntactic invariants выполнены.

NOT-INTRODUCED: external truth registered claims, novelty, priority или completeness beyond registry.

FALSIFIER-TEST: missing occurrence или hash mismatch falsifies corresponding registry assertion.

Регистратор не становится тем, что зарегистрировано.

44. Атака AN — author attribution

Строка имени автора может быть напечатана в document metadata.

RECORD-STATUS: AUTHOR-NAME-OCCURS-IN-METADATA.

ESTABLISHED: occurrence относится к exact bytes.

NOT-INTRODUCED: физическое авторство, единоличное происхождение каждой мысли или мировой приоритет.

FALSIFIER-TEST: attribution dispute не является theorem countermodel, пока не затрагивает premise.

Phase 5 не использует attribution как premise.

45. Атака АО — predecessors and adjacent works

Наличие или отсутствие citation-list не меняет proof-status theorem.

Ни внешний автор, ни школа, ни статья не являются premise.

RECORD-STATUS: PREDECESSORS-AS-PREMISES-NONE.

ESTABLISHED: dependency graph рукописи не требует внешней authority node.

NOT-INTRODUCED: отсутствие смежных работ, новизна или мировой приоритет.

NOVELTY-STATUS: NOT-ASSESSED.

PRIORITY-STATUS: NO-PRIORITY-CLAIM.

FALSIFIER-TEST: найденная сходная работа меняет historical positioning, но не refutes exact theorem без countermodel.

46. Exact falsification gate PSR-5

PSR-5 считается опровергнутой только при предъявлении всех пунктов:

1. Один exact byte-identical P .
2. Два valid exact contracts C_0, C_1 .
3. Различающиеся literal values `proof_status` из exact enum `ACCEPTED | REJECTED | UNPARSED`; различие только `terminal_witness.kind` при одном `proof_status` не активировывает PSR-5 divergence.
4. Selector z , чей полный readset доказуемо равен только P .
5. $z(P)$ корректно равен output ветви C_0 .
6. Тот же $z(P)$ корректно равен отличному output ветви C_1 .
7. Selector не читает contract прямо, косвенно, через cache, model version, filename, environment или precompiled table.

Пункты 5 и 6 требуют одному значению быть двумя различными значениями.

Если различие contracts просачивается в environment, readset не равен P .

Если outputs совпадают, separating premise отсутствует.

Если bytes различаются, это не same-surface pair.

Если один contract invalid, exact pair не проходит gate.

Следующие вещи PSR-5 не falsify:

- один accepted proof;
- один rejected proof;
- один привычный parser;
- consensus о parser;
- compiler output;
- физический результат;
- историческое происхождение;
- self-label proof;
- hash match;
- test pass.

47. Exact falsification gate ADM-5

ADM-5 считается опровергнутой одним из двух типов exact counterexample. В обоих используется generated $W_C(P) = (T, \tau)$, а не supplied certificate input.

Тип I:

1. (P, C) проходит все admission gates.
2. $Accept_C(P) = 1$ по exact definition.
3. Generated $W_C(P)$ не проходит $CertOK_C(P, W_C(P))$ либо $\tau.kind \neq TARGET - ACCEPTED$.

Тип II:

1. (P, C) проходит все admission gates.
2. Generated $W_C(P)$ проходит $CertOK_C(P, W_C(P))$ и $\tau.kind = TARGET - ACCEPTED$.
3. $Accept_C(P) = 0$ по той же exact definition.

Нельзя менять definition между пунктами.

Нельзя принимать claimed certificate без generation/recomputation.

Нельзя использовать infinite derivation при finite-derivation gate.

Нельзя вводить oracle step, отсутствующий в contract.

Нельзя подменять goal external meaning.

Следующие вещи ADM-5 не falsify:

- inconsistent axioms;
- explosion;
- proof без binding;
- иной parser;
- failing external application;
- неверный certificate;
- checker bug вне reference predicate;
- НАЗВАНИЕ доказательство.

48. Exact falsification gate NEX-5

NEX-5 считается опровергнутой только если предъявлены:

1. Один exact (P, C) .
2. Accepted-core gate: $ADM_C(P) = 1$ и `proof_status = ACCEPTED`.
3. Typed external target-space T .
4. Две admissible completions K_0, K_1 .
5. Одинаковый accepted (P, C) в обеих completions.
6. Различные target-status $\tau_0 \neq \tau_1$.
7. Selector e с полным readset только (P, C) .
8. $e(P, C) = \tau_0$ в ветви K_0 .
9. Тот же $e(P, C) = \tau_1$ в ветви K_1 .
10. Ни bridge, ни completion, ни environment marker не прочитаны selector-ом.

Если target-status не различается, separation отсутствует.

Если K записан в input, readset уже расширен.

Если selector угадывает только одну ветвь, universal correctness отсутствует.

Если внешний target не типизирован, сравнение outputs не определено.

Следующие вещи NEX-5 не falsify:

- accepted formal proof;
- soundness proof;
- работающий compiler;
- совпавшее измерение;

- летящая ракета;
- заявленная цель ракеты;
- consensus;
- успешный test suite;
- подписанный release;
- historical origin claim.

49. Implementation falsifiers

Implementation Phase 5 терпит поражение отдельно от theorem при любом из условий:

1. Parser недетерминирован при declared deterministic mode.
2. Parser читает unprotected payload как contract.
3. Accepted line не имеет axiom или earlier-rule witness.
4. Rule witness ссылается на future index.
5. rule_id witness не входит в enabled_rule_ids.
6. premise_codes witness не являются subset кодов, принятых в strict prefix.
7. conclusion_code witness не равен decoded code текущей строки.
8. Final goal не совпадает с g_C .
9. proof_status = REJECTED, но generated $W_C(P)$ проходит CertOK и TARGET-ACCEPTED.
10. proof_status = ACCEPTED, но generated $W_C(P)$ не проходит CertOK или не получает TARGET-ACCEPTED.
11. derivation_core_sha256 не совпадает с recomputed protected projection digest.
12. Claimed status принят вместо recomputation.
13. Different-contract fixture не сохраняет exact identical surface bytes.
14. Same-contract control неожиданно различает outputs.
15. External bridge прочитана как axiom.
16. proof_status = ACCEPTED сериализуется как TRUE.
17. Invalid schema приводит к unstructured acceptance.
18. Hash зависит от key order при declared canonical JSON.
19. Повторный run даёт иной verdict на exact same bytes и environment.
20. Audit пропускает обязательный contract coordinate.

Каждый implementation defect должен быть зарегистрирован своим exact status.

Implementation defect не скрывается словом warning.

Theorem не защищает implementation от ошибки.

Implementation не становится theorem authority.

50. Обязательные positive controls

Красная команда требует не только 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 проходит reference CertOK.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; proof_status = ACCEPTED; terminal_witness.kind = TARGET-ACCEPTED.

Control P9: supplied/spoofed certificate остаётся payload-inert.

Claimed certificate не является runtime input; при неизменных protected fields итог остаётся VALID-PROOF-READSET-ANALYSIS с recomputed trace и 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 formal result and split target result.

RECORD-STATUS: BINDING-COORDINATE-REQUIRED-FOR-EXPORT-STATUS.

Control P12: inconsistent explosion accepted only in explosion contract.

RECORD-STATUS: proof_status = ACCEPTED.

Control P13: payload spoof does not alter protected output.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; protected trace сохраняется, хотя artifact_sha256 может измениться.

Control P14: exact repeated run is byte-deterministic.

RECORD-STATUS: PROOF-READSET-INVARIANT-AUDIT-PASS после полного audit, включающего deterministic replay checks.

Control P15: exhaustive invariant audit covers bounded contract family.

RECORD-STATUS: PROOF-READSET-INVARIANT-AUDIT-PASS.

Ни один positive control не получает внешний truth fence.

51. Runtime fences

Каждый accepted output обязан сохранять:

```
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
```

Первый fence не отрицает machine verification syntax.

Он запрещает читать syntax checker как semantic oracle.

Второй fence ограничивает trace объявленными finite surface и contracts.

Он не объявляет полноту всех proof systems.

Третий fence не объявляет отсутствие binding, применения, цели или смысла.

Он запрещает автономный export из trace-status.

Четвёртый fence не утверждает historical origin.

Он исключает историческую строку из runtime attestation.

52. Self-application matrix

Registry class	Exact record-status	Что установлено	Что не введено
C-PUBLIC-TITLE	UNBOUNDED-STANDALONE	title bytes предъявлены	truth title
C-THEOREM-ADM-5, C-THEOREM-PSR-5, C-THEOREM-NEX-5	TEXTUAL-THEOREM-IN-DECLARED-METATHEORY	exact theorem texts предъявлены	external theorem authority
11 C-PR-* claims	VERIFIED-IN-DECLARED-ARTIFACT	bounded artifact observation совпала	referent, world truth, полнота
C-NOVELTY	NOT-ASSESSED; priority NO-PRIORITY-CLAIM	отсутствие novelty/priority attestation в registry	novelty или priority

Для всех 16 claims semantic status равен NOT-MACHINE-VERIFIED.

Matrix применяется без исключения к exact frozen registry Phase 5. Ни один из этих 16 claim records не получает status шире указанного; statement обо всех release artifacts self-audit не делает.

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.

Любой другой id отсутствует в frozen self-audit input и не получает status от Phase 5 self-audit engine.

Запрещённые unbounded claims:

математика никогда не работает.

ни одной истины не существует.

никакого количества не существует.

все доказательства ложны.

ракета не летит.

применения невозможны.

смысла не существует.

Phase 5 впервые в мире это доказала.

Запрет не смягчает главный угол.

Он не позволяет заменить точный разрез новой неподтверждённой строкой.

54. Результат красной команды

Все strongest objections оставляют один и тот же остаток.

Parser objection показывает parser-coordinate.

Axiom objection показывает axiom-coordinate.

Rule objection показывает rule-coordinate.

Goal objection показывает goal-coordinate.

Proof-assistant objection показывает kernel/environment-coordinate.

Compiler objection показывает toolchain-coordinate.

Soundness objection показывает meta-contract.

Measurement objection показывает measurement bridge.

Rocket objection показывает application, observation и purpose bridges.

Historical objection не входит в premise.

Self-label objection остаётся payload.

Test objection остаётся test record.

Hash objection остаётся byte relation.

Ни одна атака не предъявила selector, способный получить два различных correct outputs из одного и того же полного readset.

Ни одна атака не предъявила mismatch между accepted finite derivation и CertOK её generated trace с TARGET-ACCEPTED.

Ни одна атака не предъявила external target-status без bridge при separating completions.

Эти строки являются итогом declared adversarial set.

Они не объявляют исчерпание всех возможных будущих атак.

55. Поверхностные, digest- и registry-атаки

55.1. CRLF, LF и final newline

Surface identity задаётся exact bytes либо exact declared normalization.

Файл с CRLF и файл с LF не называются byte-identical.

Файл с final LF и файл без final LF не называются byte-identical.

Если normalizer сводит их к одному surface, normalizer и его version входят в contract.

RECORD-STATUS: DISTINCT-SURFACE-BYTES до declared normalization.

RECORD-STATUS: SAME-NORMALIZED-SURFACE-UNDER-C после declared normalization.

ESTABLISHED: identity criterion различает два уровня.

NOT-INTRODUCED: semantic equivalence или внешний referent.

FALSIFIER-TEST: нельзя заявлять same-bytes pair после скрытой newline conversion.

55.2. Empty final line

В current runtime каждая reached supplied line, включая line_text == "", обязана иметь exact decoding item; missing mapping на reached index даёт UNPARSED. Если earlier mapped line уже завершила run как REJECTED, later line не достигается. Runtime не имеет ignored-empty-line mode.

Ignored token, formula separator или malformed-line policy возможны только в явно расширенном parser supertype. Такое расширение добавляет protected parser-mode coordinate и не является executable result base checker.

RECORD-STATUS: в base runtime определяется first-terminal traversal и exact decoding каждой reached supplied line; в extension — отдельно объявленным parser-mode.

ESTABLISHED: current empty-line decoding policy входит в parsing readset.

NOT-INTRODUCED: пустота как объект или отсутствие чего-либо вне surface.

FALSIFIER-TEST: parser-blind selector должен разделить outputs на одном exact surface.

55.3. Unicode homoglyph arrow

Знак → и sequence -> имеют разные code points.

Знак -> с Unicode minus содержит ещё одну sequence.

Визуальное сходство не является byte equality.

Parser может объявить normalization map.

Тогда map входит в contract.

RECORD-STATUS: DISTINCT-CODEPOINT-SURFACES без map.

ESTABLISHED: glyph appearance не подменяет codepoint sequence.

NOT-INTRODUCED: единый implication-object за тремя начертаниями.

FALSIFIER-TEST: same-bytes fixture обязан сравнивать bytes, а не screenshot.

55.4. Unicode whitespace

ASCII space, non-breaking space и thin space различны как code points.

Parser-policy может склеить их.

Склейка является operation contract.

RECORD-STATUS: DISTINCT-SURFACE-BYTES до operation.

ESTABLISHED: exact surface audit видит различие.

NOT-INTRODUCED: различие meanings.

FALSIFIER-TEST: hidden whitespace canonicalization снимает claim byte-identity, если она не предъявлена.

55.5. Strict-prefix premise attack

Допустимый premise index должен удовлетворять $j < i$.

Условие $j \leq i$ ошибочно допускает self-premise.

Условие $j \text{ exists}$ ошибочно допускает future premise.

RECORD-STATUS: VALID-PROOF-READSET-ANALYSIS; runtime не читает claimed premise indices и генерирует только strict-prefix premise_witnesses.

ESTABLISHED: claimed graph не проходит strict-prefix gate; runtime генерирует собственные witnesses.

NOT-INTRODUCED: невозможность cyclic proof в любом другом contract.

FALSIFIER-TEST: reference checker, принявший $j=i$ при declared strict-prefix mode, терпит implementation defeat.

55.6. Goal встретился раньше, но финал иной

Surface:

```
goal
aux
```

Обе строки могут быть individually admitted.

Goal-code встречается на индексе 0.

Но mode требует final-line goal match.

Final line равна aux.

RECORD-STATUS: proof_status = REJECTED; terminal_witness.kind = TARGET-LINE-MISMATCH.

ESTABLISHED: occurrence goal раньше финала не заменяет final witness.

NOT-INTRODUCED: goal achievement, внешняя цель или смысл.

FALSIFIER-TEST: checker, принимающий этот record при final-line mode, falsifies implementation claim.

55.7. Минимальная PSR-пара

Для сильнейшего pair-control фиксируются одинаковыми:

- exact surface bytes;
- alphabet;
- parser и parser version;
- formula sort;

- axioms;
- полный каталог rule definitions;
- goal;
- mode;
- payload;
- metadata.

Различается только protected coordinate `enabled_rule_ids`.

В C_0 modus-ponens rule id включён.

В C_1 тот же id выключен.

RECORD-STATUS(C_0): `proof_status = ACCEPTED`.

RECORD-STATUS(C_1): `proof_status = REJECTED`.

ESTABLISHED: одной различающей coordinate достаточно для PSR pair.

NOT-INTRODUCED: всеобщая зависимость каждой математической практики от одного конкретного JSON field.

FALSIFIER-TEST: selector должен не читать `enabled_rule_ids` и всё же вернуть оба separated outputs.

55.8. Artifact hash и derivation-core digest

В namespace `proof_readset_checker artifact_sha256` относится к canonical digest parsed artifact-value по declared canonicalization.

Он не заявлен digest raw file bytes, CRLF presentation или произвольной сериализации того же JSON value.

В namespace `phase5_self_audit` field с тем же spelling `artifact_sha256` относится, напротив, к exact raw fixture file bytes и замораживает файл перед checker call. Это разные record contracts. Равенство между двумя digest не требуется и не заявлено.

`surface_sha256` относится к protected record `{surface_id, lines}`.

`derivation_core_sha256` относится только к declared protected derivation core: `surface_sha256`, `decoding`, `axiom_codes`, полный `normalized rules[{rule_id,premise_codes,conclusion_code}]` catalog без per-rule payload, `enabled_rule_ids`, `target_code`.

Эти digest scopes нельзя склеивать.

Изменение payload может изменить `artifact_sha256` без изменения `surface_sha256` и `derivation_core_sha256`.

Изменение `protected rule coordinate` меняет `canonical core object`, после чего `checker` пересчитывает `derivation_core_sha256`. Равенство SHA-256 для разных `core objects` означало бы collision; injectivity hash model не утверждается. Для `frozen minimal pair` фактически записаны разные recomputed digests.

RECORD-STATUS: `equality proof-readset artifact_sha256`, `surface_sha256` и `derivation_core_sha256` — три разные record relations с разными scopes; `self-audit raw-file artifact_sha256` образует четвёртый, отдельно qualified scope.

ESTABLISHED: каждый digest фиксирует свой declared byte scope.

NOT-INTRODUCED: truth derivation, identity scopes или authority hash.

FALSIFIER-TEST: scope confusion falsifies integrity implementation claim.

55.9. Duplicate claim object в self-audit input

Claim registry требует exact declared multiplicity объектов внутри `self_claims_phase5.json`.

Два claim objects с одним id при требовании одного не проходят audit.

RECORD-STATUS: SELF-AUDIT-DUPLICATE-CLAIM.

ESTABLISHED: occurrence count расходится с registry.

NOT-INTRODUCED: ложность claim-content.

FALSIFIER-TEST: audit, вернувший PASS при duplicate exact claim, терпит implementation defeat.

55.10. Extra unregistered claim object

Появление нового bounded claim object внутри `self_claims_phase5.json` требует frozen registry id. Engine не сканирует весь текст файлов 00–05 и не заявляет, что обнаруживает каждое предложение, которое reader мог бы назвать claim.

RECORD-STATUS: SELF-AUDIT-EXTRA-CLAIM.

ESTABLISHED: exact registry input содержит extra object/id.

NOT-INTRODUCED: truth или falsehood extra claim.

FALSIFIER-TEST: PASS при forbidden unregistered claim falsifies audit implementation.

55.11. Nested prohibited string

Запрещённая строка может быть спрятана в nested JSON value, array, comment-like payload или escaped sequence внутри protected `self_claims_phase5.json` tree.

Self-audit обязан обходить весь declared self-claim input tree, а не только top-level keys. Эта проверка не является свободным scan всех manuscript sentences.

RECORD-STATUS: SELF-AUDIT-PROHIBITED-STRING-FOUND.

ESTABLISHED: forbidden occurrence присутствует в protected audit scope.

NOT-INTRODUCED: внешний referent этой строки.

FALSIFIER-TEST: audit, проверяющий только top level и возвращающий PASS, терпит implementation defeat.

55.12. Явная область pair theorem

PSR-5 квантифицирует по exact pair с separating outputs.

Она не квантифицирует одним шагом по всей математике.

Она не утверждает, что все mathematical surfaces имеют разные statuses во всех contracts.

Она не утверждает, что ни один contract не может быть широко разделяемым.

Она устанавливает невозможность contract-blind selector на предъявленной separating pair и на любой другой pair той же формы.

RECORD-STATUS: PAIR-RELATIVE-READSET-THEOREM.

ESTABLISHED: necessity различающей coordinate при separated outputs.

NOT-INTRODUCED: historical, sociological или ontological universal о всей математике.

FALSIFIER-TEST: предъявить selector-only counterexample на exact separating pair; встречное обобщение о всей математике не адресует pair premise.

Эта граница не смягчает knife.

Даже один accepted proof не получает автономный статус из собственных линий, если существует exact contract-coordinate, без которой его status не определён.

56. Финальный нож Phase 5

Доказательство не находится в начертании.

Оно является статусом начертания под контрактом.

И этот статус не предъявляет доказанное вне контракта.

Сначала предъявлены линии.

Статус доказательства появляется только после контракта их чтения.

Допуск внутри контракта не предъявляет названное.

Работающий checker устанавливает checker-output.

Работающий compiler устанавливает compiler-output.

Совпавшее измерение устанавливает match по записанному protocol.

Летающая ракета устанавливает различие по записанному observation contract.

Ответ на для чего появляется только как новая purpose-coordinate.

Ни title, ни theorem-label, ни тест, ни hash не освобождены от этого разреза.

PREDECESSORS-AS-PREMISES: NONE.

NOVELTY-STATUS: NOT-ASSESSED.

PRIORITY-STATUS: NO-PRIORITY-CLAIM.

Имя не предъявляет названное.

Формула не предъявляет истину.

Допущенный вывод не предъявляет доказанное.