
ЦИКЛ НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ

Теорема наименьшего происхождения для циклических записей и граница
самореференции

Кирилл Шебетов · beforeword / research

Phase 3 · technical version 3.0.0 · 28 августа 2026

Содержание

Аннотация

0. Нулевая уступка словам
1. Точный объект опровержения
2. Линия Phase 0 → Phase 1 → Phase 2 → Phase 3
3. Конечный типизированный граф с циклами
4. Оператор переноса и наименьшая неподвижная точка
5. CP-LFP-3 (CFP-3): теорема наименьшего происхождения
6. Strongly connected components
7. Самопетля без seed
8. Взаимный цикл без seed
9. Fixed point не равен выбранному fixed point
10. SG-LFP-3 (GST-3): grounded support
11. Циклический provenance cut
12. Композиция с AST-1
13. Самореференция не получает исключения
14. Математическое чтение как вход
15. Максимальная уступка техническому успеху
16. Вопрос зачем
17. Итерация, длительность и предел
18. Сильнейшие возражения
19. Точные фальсификаторы
20. Исполняемая область
21. Самоприменение
22. Предшественники и статус новизны
23. Граница притязаний
24. Итог

Статус источников

CP-LFP-3 (CFP-3): теорема наименьшего происхождения для циклической записи

0. Статус обозначений
1. Место относительно Phase 0–2
2. Конечная циклическая запись
3. Оператор циклического происхождения
4. CP-LFP-3: теорема пути
5. Цикл не является источником
6. SCC-конденсация
7. Разрез в циклическом графе
8. Композиция с AST-1
9. Полное замыкание не создаёт внешний target

10. Ковариантность переименования
11. Снимки и точная граница неретроактивности
12. Admission gates
13. Нормативные статусы
14. Обязательные контроли
15. Точные фальсификаторы
16. Самоприменение и максимальная строка

SG-LFP-3 (GST-3): теорема опоры и исходных введений

0. Нормативный статус
1. Зачем нужен AND/OR-слой
2. Конечный support-contract
3. Наименьшее AND/OR-замыкание
4. Конечное support tree
5. SG-LFP-3: теорема support tree
6. Минимальные root bases
7. Точное блокирование root
8. Циклы в AND/OR-hypergraph
9. CP-LFP-3 как unary частный случай
10. Binding, event, cause, application, purpose и meaning
11. Self-reference
12. Admission gates
13. Нормативные статусы
14. Обязательные контроли
15. Точные фальсификаторы
16. Максимальная формулировка

FP-SEL-3 (FPS-3): выбор неподвижной точки и круговой остаток

0. Нормативный статус
1. Базовый положительный контракт
2. Интервал fixed points
3. Support-grounded fixed point
4. Unsupported circular residue
5. Минимальные примеры
6. Выбор fixed point как вход
7. Target selection после fixed point
8. Self-reference и quotation
9. Feedback, controller и ракета
10. Admission gates
11. Нормативные статусы
12. Обязательные контроли
13. Точные фальсификаторы

14. Максимальная формулировка

САМОДОКАЗАТЕЛЬСТВО НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ

Самоссылка, взаимная проверка, математическое чтение, ракета и вопрос зачем

Аннотация

0. Сначала предъявлены строки

1. Отношения, которые нельзя склеивать

2. Точная линия Phase 0 → Phase 1 → Phase 2 → Phase 3

3. Один узел, объявляющий себя доказанным

4. Самопетля с явным seed

5. Два узла, проверяющие друг друга

6. Кольцо из множества проверяющих

7. Support graph: ссылка не равна основанию

8. Proof-checker, проверяющий собственный код

9. Самодоказательство и уровни claim

10. Fixed-point policy как часть proof contract

11. Математика начинается в этом record как reading contract

12. Внутренняя семантика допускается полностью

13. Максимальная уступка ракете

14. Feedback loop ракеты

15. Что именно доказывает успешный полёт

16. Вопрос зачем после полной уступки

17. Сто, двести, триста обходов

18. Constitutive convention

19. Coinduction

20. Mutual verification и независимость

21. Self-modification и меняющийся graph

22. Самые сильные возражения

23. Точные application-falsifiers

24. Самоприменение этого файла

25. Граница application-притязаний

26. Итог

Статус источников

05. ВРАЖДЕБНАЯ ПРОВЕРКА И САМОПРИМЕНЕНИЕ PHASE 3**ЦИКЛ НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ**

0. Назначение и неподвижная граница

1. Точные объекты атаки

2. Красная команда против CFP-3

3. Красная команда против GST-3

4. Красная команда против выбора fixed point

5. Скрытый readset, незакрытая цепь и регресс

6. Payload spoof и подмена CP-sort / SG-role
7. Максимальная уступка: ракета и успешное применение
8. Перформативное противоречие и самоприменение
9. Условия несостоятельности
10. Обязательные положительные и отрицательные controls
11. Точный реестр claim / status / falsifier
12. Финальная граница красной команды

Аннотация

Phase 2 локализовала различающий typed atom среди предков заключения, но сделала это для конечного ациклического графа. Настоящее исследование снимает именно это ограничение. Разрешаются конечные направленные графы с самопетлями, взаимными ссылками и strongly connected components. Слова цикл, источник, истина, доказательство, причина, цель и смысл не принимаются за то, что ими названо. Они входят только как точные записи и технические коды объявленного контракта.

Для каждого узла задаются защищённые origin-tagged introductions. На полном решётчатом пространстве provenance-состояний определяется монотонный оператор переноса: узел содержит собственные introductions и всё, что поступило по записанным входящим рёбрам. Provenance выбирается не произвольной неподвижной точкой, а наименьшей неподвижной точкой, полученной итерацией от пустого состояния.

Доказывается **CP-LFP-3 (CFP-3)** — теорема циклического provenance и путей. Origin-tagged atom входит в наименьшее provenance узла тогда и только тогда, когда atom защищённо введён в некотором узле-источнике и от этого узла существует конечный направленный путь к рассматриваемому узлу. Цикл может многократно переносить введённый atom, но source-free strongly connected component не порождает его одной циркуляцией.

Для конечных положительных AND/OR-hypergraphs после нормализации nullary constructors, с непустыми premises у ordinary rules, доказывается **SG-LFP-3 (GST-3)**: каждое заключение в наименьшем замыкании имеет конечное дерево поддержки, листья которого являются объявленными seeds. Чистая самоподдержка или взаимная поддержка без protected seed и без альтернативного конечного seed-grounded support tree не входит в μT_S . Альтернативные минимальные bases сохраняются как точные множества записанных оснований.

Наконец, **FP-SEL-3 (FPS-3)** отделяет решение уравнения неподвижной точки от выбора конкретной неподвижной точки. Для support-оператора $T_S: X \in \text{Fix}(T_S)$ является support-grounded тогда и только тогда, когда $X = \mu T_S$. Если допустимо несколько fixed points, правило least, greatest, stable или любое иное правило выбора является дополнительным входом; при $X \neq \mu T_S$ каждый code из непустого residue $X \dot{\vdash} T_S$ не имеет конечного seed-grounded support tree.

Результат применяется к самодоказательству, взаимной проверке, рекурсивным определениям, математическому reading contract, техническому применению и вопросу зачем. Он не объявляет циклы бесполезными и не отрицает условную силу рекурсивной математики. Он полностью

закрывает более узкое притязание: замкнутая запись сама по себе не становится первым источником отсутствующего protected atom.

Ключевые слова: граница записи; циклическое provenance; least fixed point; strongly connected component; origin tag; grounded support; самореференция; взаимная проверка; fixed-point policy; application binding; цель; self-application.

0. Нулевая уступка словам

Сначала предъявлены строки:

```
цикл
самоссылка
истина
доказательство
проверено
ракета
событие
цель
смысл
```

Ни одна строка не предъявляет названное. Строка `цикл` не образует цикл; цикл появляется только после объявления множества узлов и точных рёбер. Строка `истина` не получает truth-status. Строка `доказательство` не является proof-object. Строка `ракета` остаётся строкой, даже если вне этого файла принимается максимальная уступка, что ракета существует и летит.

Те же ограничения относятся к формальным словам настоящего исследования. Graph, atom, origin, fixed point, least, support и source являются метками ролей. Их написание не устанавливает правильность модели, соответствие чему-либо вне записи или приоритет исследования.

На нулевом уровне фиксируется exact carrier. Затем отдельный reading contract вводит:

- узлы;
- направленные рёбра;
- типы atom;
- защищённые поля introduction;
- правило переноса;
- fixed-point policy;
- критерий принятия результата.

До этого есть carrier. После этого есть запись, читаемая по объявленному контракту. Обратный перенос запрещён: более поздняя метка источник не превращает более ранний payload в источник.

1. Точный объект опровержения

Phase 3 не использует безграничную фразу любой цикл ничего не создаёт. Она ложна уже внутри record-контракта: узел цикла может иметь local introduction, цикл может получить atom по входящему ребру, а рекурсивная процедура может корректно вычислить новый payload из введённых данных.

Опровергается точное притязание **cyclic self-origin**:

protected atom входит в least provenance узла только потому, что записи циклически ссылаются друг на друга, хотя ни один достигающий узел не имеет защищённого introduction этого atom.

Для наименьшей монотонной semantics это притязание имеет точный отрицательный ответ. Если atom присутствует, существует origin-tagged introduction и конечный путь от origin к месту присутствия. Если такого introduction нет, циркуляция не заполняет пустое provenance.

Слово полностью далее относится к универсальному квантору по всем конечным направленным графам и всем atom, удовлетворяющим объявленному positive-union contract. Оно не относится к непредъявленному классу всех возможных употреблений слов цикл, математика ИЛИ истина.

2. Линия Phase 0 → Phase 1 → Phase 2 → Phase 3

Четыре фазы не повторяют один тезис. Каждая закрывает точный остаток предыдущей.

Phase 0: bridge provenance

MRB-0 разделила формальный тип M , прикладной тип A и целевой тип V . Единственное правило введения A требовало bridge, а зависимости объединялись и не удалялись. Поэтому принятый proof-object типа A не мог иметь пустое множество bridge dependencies.

Остаток: результат зависел от конкретной типовой архитектуры и требовал отдельного переноса t на иной корпус.

Phase 1: функциональная граница

RBI-1 заменила архитектурный запрет общей теоремой факторизации. Для полного доступного входа p и независимо заданного target Q :

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

Одна пара с одинаковым полным input и различным target исключала любой evaluator этого input. Более длинный расчёт, внутренняя модель, oracle-answer, случайность или transcript помогали только тогда, когда вносили различающий канал.

Остаток: было установлено, что различие добавлено, но не было локализовано место его введения.

Phase 2: предковая локализация

AST-1 ввела origin-tagged ancestry в конечном immutable DAG. Если корректное заключение имеет вид

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

то

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

Следовательно, различающий typed atom имеет origin среди рефлексивных предков заключения. FDT-2 отдельно локализовала первый selectable stage конечной фильтрации.

Остаток: DAG запрещал самопетли и взаимные ссылки. Обычное слово предок переставало различать узлы внутри strongly connected component. Phase 2 прямо оставила циклы отдельной областью, требующей fixed-point semantics.

Phase 3: происхождение внутри циклов

Phase 3 сохраняет origin tags, но заменяет рекурсивный ancestor-union на наименьшую неподвижную точку монотонного оператора. Strongly connected component может циркулировать atom, однако его origin остаётся узлом защищённого introduction. Если допустимы несколько fixed points, выбор одного из них выводится из тени и помещается в полный readset.

Итоговая линия:

```

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

```

Это усиление области, а не расширение слов к названному ими.

3. Конечный типизированный граф с циклами

Пусть

$$G = (V, E)$$

— конечный направленный граф. Ребро

$$u \rightarrow v$$

означает только записанную dependency: узел v объявил u входом переноса. Оно не означает физическую причинность, временное предшествование, доверие или истинность.

Разрешены:

- самопетли $v \rightarrow v$;
- взаимные рёбра $u \rightarrow v \rightarrow u$;
- произвольные конечные strongly connected components;
- ациклические участки между компонентами.

Сорта Phase 3:

$$T_3 = \{ \underset{\text{f}}{\text{FORMAL}}, \underset{\text{b}}{\text{BINDING}}, \underset{\text{e}}{\text{EVENT}}, \underset{\text{g}}{\text{GOAL}},$$

В формулах это uppercase mathematical sort-codes. Поле JSON sort сериализует их фиксированной биекцией $\text{FORMAL} \leftrightarrow \text{"formal"}$, $\text{BINDING} \leftrightarrow \text{"binding"}$, $\text{EVENT} \leftrightarrow \text{"event"}$, $\text{GOAL} \leftrightarrow \text{"goal"}$, $\text{REFERENT} \leftrightarrow \text{"referent"}$, $\text{CAUSE} \leftrightarrow \text{"cause"}$, $\text{APPLICATION} \leftrightarrow \text{"application"}$, $\text{MEANING} \leftrightarrow \text{"meaning"}$. Верхнеуровневое поле узла kind использует отдельные uppercase literals и не является полем sort.

Имена сортов не устанавливают соответствующие отношения. EVENT — сорт event-code; не событие. REFERENT — сорт referent-code; BINDING остаётся отдельным сортом. MEANING — сорт meaning-code; не смысл.

Для каждого $\tau \in T_3$ фиксируется конечное множество допустимых atom-id A_τ . Для узла v защищённое локальное введение задаётся множеством

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

Payload инертен. Если в payload написано goal, true, proved, rocket или полная сериализация atom, это не создаёт introduction. Защищённое поле introduction и payload являются разными координатами record.

Origin-tagged universe:

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

Локальный набор введений узла:

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

Tag origin сохраняется при переносе. Один и тот же atom-code, введённый в двух узлах, образует две разные origin-tagged записи.

4. Оператор переноса и наименьшая неподвижная точка

Provenance-состояние — семейство

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

Порядок на L задаётся покоординатным включением. Нижний элемент:

$$\perp_v = \emptyset.$$

Оператор переноса:

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

Он монотонен:

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

Итерация начинается не с желаемого результата, а с пустого состояния:

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

Так как V и Ω конечны в объявленном artifact-contract, возрастающая последовательность стабилизируется после конечного числа строгих добавлений. Определяется

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

Слово *least* не является риторической оценкой. Оно означает минимальный по включению fixed point оператора Φ_G . Эта policy записана до вычисления результата.

5. CP-LFP-3 (CFP-3): теорема наименьшего происхождения

Для любых $s, v \in V, \tau \in T_3$ и $a \in A_\tau$:

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

Здесь $s \rightsquigarrow v$ — конечный направленный путь, включая путь длины ноль при $s = v$.

Доказательная схема

После первой итерации присутствуют только local introductions. После второй — introductions, дошедшие по пути длины не более одного ребра. Индукция по n даёт:

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

ровно тогда, когда a введён в s и существует путь $s \rightsquigarrow v$ длины не более n . Объединение всех конечных итераций даёт прямое направление теоремы. Обратное направление следует прохождением любого конечного пути: introduction входит в источник и переносится по каждому ребру.

Полное формальное доказательство, леммы монотонности и условия контракта вынесены в отдельный теоремный слой Phase 3. Здесь фиксируется центральное следствие:

*Наличие atom в P_v^μ имеет два обязательных поля: introduction и конечный путь переноса.
Цикл сам не заменяет ни одно из них.*

6. Strongly connected components

Пусть $SCC(v)$ — strongly connected component узла v . Сжатие каждой SCC в одну вершину образует condensation graph

$$Cond(G),$$

который является DAG.

CP-LFP-3 даёт точное описание provenance компоненты:

1. atom может быть локально введён внутри SCC;
2. atom может войти по пути из upstream-компоненты;
3. после входа atom может распространиться по достижимым узлам SCC;
4. отсутствие локального и upstream introduction оставляет atom отсутствующим во всей SCC.

Поэтому фраза в цикле нет первого узла не создаёт исключения. Хронологически первого узла внутри замкнутой записи может не быть в модели, но origin atom — не первый по кругу. Origin — узел защищённого introduction. Его тег сохраняется после любого числа обходов.

Обозначим $s \rightsquigarrow C$ тогда и только тогда, когда существует $v \in C$ с $s \rightsquigarrow v$. Точное SCC-следствие:

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

Source-free component может быть структурно сложной, иметь тысячи рёбер и бесконечное число описываемых обходов. Сложность маршрута не является introduction.

7. Самопетля без seed

Рассмотрим один узел v и ребро $v \rightarrow v$. Пусть

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

Тогда итерация снизу даёт:

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

и

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

Запись может содержать payload:

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

Payload не изменяет результат. Чтобы atom `VALID` вошёл в P_v^μ , нужен local либо upstream introduction. Другая fixed-point policy может выбрать другое состояние, но не изменяет $\mu\Phi_G$ прежнего instance.

Это не утверждение, что самореферентные формальные объекты запрещены. Установлено только отсутствие origin в данном least-provenance contract.

8. Взаимный цикл без seed

Пусть

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

и оба local introduction sets пусты. Фразы

```
u: v проверен
v: u проверен
```

могут находиться в payload. Наименьшая итерация всё равно начинается с пустого состояния и остаётся пустой.

Если же u защищённо вводит atom a , то

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

Цикл теперь переносит защищённо введённый atom. Он не скрывает его origin: тег u сохраняется в обоих узлах.

Отсюда полярная симметрия:

- cycle без local/upstream introduction не получает atom;
- cycle с local/upstream introduction законно распространяет atom;
- наличие introduction не превращает atom-code в референт, событие или истину.

9. Fixed point не равен выбранному fixed point

Множество неподвижных точек:

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

Уравнение может иметь больше одного решения. Для самопетли без introduction оно принимает форму

$$X_v = X_v.$$

Пустое состояние удовлетворяет этой записи; если $\Omega \neq \emptyset$, ей могут удовлетворять и непустые состояния. Само равенство не выбирает одно из них.

Контракт, который выбирает одну fixed point как accepted state, обязан указать поле

`fixed_point_policy`

Например:

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

Эти строки являются policy-codes. Их написание не доказывает правильность policy, но без них readset контракта выбора неполон. В CP-LFP-3 policy заранее зафиксирована как least-from-bottom.

Unary path-grounded следствие и FP-SEL-3

Для оператора positive-union типа Φ_G единственная fixed point, в которой каждое membership имеет конечный introduction-to-node path witness, равна $\mu \Phi_G$. Это unary path-grounded следствие CP-LFP-3.

Следовательно, если

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

то существует circular residue

$$X \{ P \overset{\mu}{\hookrightarrow}$$

без finite-path introduction witness. В конечном графе обратное прослеживание dependency достигает cycle внутри unsupported residue.

Нормативная **FP-SEL-3** относится к support-оператору T_S , определённого в следующем разделе:

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

Если $X \in \text{Fix}(T_S)$ и $X \neq \mu T_S$, то каждый $q \in X \dot{\cup} T_S$ не имеет конечного seed-grounded support tree. Наличие других seeds этого не меняет: например, правило $\{q, s\} \Rightarrow q$ при $s \in S$ сохраняет обязательную circular branch для q .

Это не делает nonleast fixed point ложной. Она может быть принята coinductive или иной policy. Точный вывод иной:

policy, разрешившая unsupported residue, является дополнительным входом; residue не был введён одной замкнутостью equations.

10. SG-LFP-3 (GST-3): grounded support

Обычный directed graph переносит отдельные atom. Для правил с несколькими premises используется конечный монотонный AND/OR-hypergraph. В абстрактной теореме фиксированы конечное множество code-id N , seed-set $S \subseteq N$ и конечное множество правил R . Для каждого $r \in R$:

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

Оператор равен

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

В executable refinement namespace bases и namespace claims разделены tagged-вложением

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

Здесь B содержит tagged base_id, а C — tagged claim_id. Для inference множество $\text{Prem}(r)$ является объединением required_base_ids из B и premise_claim_ids из C , а $\text{Head}(r) \in C$ задаётся conclusion_id. Поэтому executable checker сообщает claim-проекцию абстрактного least closure, не стирая различие BASE/CLAIM при проверке.

Условие $\text{Prem}(r) \neq \emptyset$ обязательно. Обычный empty-premise rule находится вне SG-контракта. Nullary constructor до сериализации нормализуется в отдельный protected base, например RULE-SEED:<rule-id>, а inference к conclusion явно перечисляет этот base_id в required_base_ids. Если одновременно пусты premise_claim_ids и required_base_ids, checker не выполняет нормализацию за автора, а возвращает INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT с reason-code NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE.

SG-LFP-3 устанавливает:

каждый root-code в least closure имеет конечное дерево поддержки; внутренние вершины соответствуют объявленным rules, а каждый лист является объявленным seed.

Самопетля claims

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

без rule-path от S даёт $C : q \notin \mu T_S$. Взаимная пара claims

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

без rule-path от S даёт $C : p, C : q \notin \mu T_S$. Положительный executable-совместимый control использует отдельный base:

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

Тогда оба claim-code входят в least closure, а каждое предъявленное support tree заканчивается leaf $B : b$, не claim-code $C : p$.

Если root имеет несколько альтернативных минимальных seed-bases, они образуют antichain по включению. Выбор одного удобного basis не стирает остальные и не делает root безусловным.

SG-LFP-3 относится к support-records. Слово premise не предъявляет факт; слово proof не предъявляет названную истину.

11. Циклический provenance cut

Обозначим untagged projection $Prov_H^\mu(x) = \{(\tau, a) : \exists s(\tau, a) \in P_H^\mu(x)\}$. Пусть фиксированы root r , untagged code (τ, a) и множество его достигающих root introducers

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

Через $P_G(s, r)$ обозначается множество конечных направленных путей из s в r , через $V(p)$ — множество вершин пути p .

В internal-cut contract все узлы $S_{\tau,a}^r$ и root защищены. Разрешено удалить

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

без rewiring, после чего least provenance пересчитывается на $G - B$. Тогда:

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

Если $S_{\tau,a}^r = \emptyset$, cut-status равен ABSENT, а rank равен null: пустой hitting set не выдаётся за cut-certificate. Если introducer совпадает с root или соединён с ним прямым ребром, допустимого internal cut нет; это UNBLOCKABLE-PROTECTED-ENDPOINT, а не cut rank zero. Во всех остальных случаях exact минимум получает CUT-CERTIFIED. Циклы не отменяют критерий: любой достигающий root walk

содержит простой путь. Минимальный combined internal cut измеряет только структурную избыточность записанных маршрутов; он не измеряет истинность, важность, физическую причинность или смысл.

12. Композиция с AST-1

Пусть $K \neq \emptyset$ — объявленный класс exact-record индексов. Для заключения v определим циклическую ancestry-проекцию:

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

Пусть ρ содержит полный readset evaluator, F тотальна и для каждого $k \in K$

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

Тогда функциональная теорема Phase 1 остаётся неизменной:

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

Если $\rho(k_0) = \rho(k_1)$, но $Q(k_0) \neq Q(k_1)$, то $\eta_v^\mu(k_0) \neq \eta_v^\mu(k_1)$. По CP-LFP-3 различающий origin-tagged member этой проекции имеет защищённое introduction и путь к заключению.

Если используется другая fixed-point policy, её идентификатор, порядок, admissibility rules и selector включаются в ρ . Скрытая замена least на greatest является изменением входа, а не опровержением theorem.

13. Самореференция не получает исключения

Строка

эта строка доказана этой строкой

может быть узлом graph. Самопетля может быть записана. Parser может принять её форму. Checker может классифицировать её по заранее заданному recursive contract.

Из этого не следует protected atom PROVED, если:

- он не введён локально;
- он не поступил по пути от другого introduction;
- least closure не имеет seed-support tree;
- policy не содержит отдельного правила, учреждающего такой статус.

Если правило учреждения записано, оно является новым input. Тогда status может быть корректным **внутри этого контракта**. Phase 3 не запрещает учреждённые статусы; она фиксирует их зависимость.

Взаимная фраза А подтверждает В, В подтверждает А также не образует независимого основания. При наличии доверенного seed, kernel-rule, certificate-root или external attestation цикл может переносить введённый status. Origin остаётся в protected introduction или seed; policy отдельно остаётся в readset и может выбрать nonleast state.

Подробное применение дано в 04_APPLICATION_SELF_PROOF_RU.md.

14. Математическое чтение как вход

Структурный порядок сохраняется:

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

Слово математика не появляется на нулевом уровне. Carrier получает status READ-AS-MATHEMATICAL после parser-, syntax- и rule-records. Если эти records циклически ссылаются на свои результаты, CP-LFP-3 показывает происхождение protected atoms, а FP-SEL-3 показывает роль policy.

Формальная система может корректно доказать теорему о собственной записи, checker или semantics. Этот результат сохраняется как

FORMAL - STATUS - IN - DECLARED - CONTRACT

и не переименовывается в

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

без отдельных relations и inputs.

Фраза математика возникла из языка не используется как историческая посылка. Установлен только порядок доступа в предоставленном record: mathematical reading требует reading contract и поэтому не служит основанием собственного появления в этом же контракте.

15. Максимальная уступка техническому успеху

Для применения допускается одновременно:

- ракета существует;
- ракета построена;
- formal calculation корректен;

- calculation использован при проектировании и управлении;
- controller, telemetry и checking chain сработали;
- траектория совпала с расчётом по принятому criterion;
- ракета летит.

Эти пункты принимаются как максимальная уступка, а не выводятся из строк настоящей рукописи.

Даже в этой уступке различаются records:

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

Циклическая цепь $\text{simulation} \leftrightarrow \text{controller} \leftrightarrow \text{telemetry} \leftrightarrow \text{model}$ может быть технически полезной и иметь взаимные зависимости. event, fit и application здесь являются atom/claim-role codes, каждому из которых отдельно назначается один из восьми сортов T_3 ; новых неявных сортов они не вводят. Соответствующие protected atoms входят через sensor-, binding-, criterion- и protocol-records. Циркуляция распространяет их; она не помещает их задним числом в ранний formula record.

Успех не опровергается. Опровергается обратный перенос успеха в прежний carrier.

16. Вопрос зачем

Циклическое повторение goal-строки не является её justification. Различаются:

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

Эти labels являются claim-role codes, а не дополнительными сортами сверх T_3 . Каждый status получает отдельно объявленные sort, introduction rule и support bases. Goal может быть protected seed для оптимизации. Оптимизация может корректно вычислить trajectory относительно goal. Она не выбирает, почему goal должен быть принят.

Если единственные rules с heads $C:g$ и $C:d$ образуют взаимную пару

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

и ни один из этих claims не имеет rule-path от S , то

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

В S могут находиться unrelated bases, а из них могут выводиться другие claims; утверждение исключает только $C:g$ и $C:d$ при указанном отсутствии пути поддержки. Если policy учреждает coinductive acceptance, policy является отдельным input. Если goal вводится через protected base и

отдельное inference, его exact supporting base/base_id явно предъявлен; одно перечисление goal как claim-premise его не вводит.

Поэтому максимальная уступка ракете не закрывает за чем одним техническим кругом. Она позволяет записать успешную причинно-техническую цепь по принятому контракту; goal, endorsement и justification остаются отдельными координатами.

17. Итерация, длительность и предел

На конечном graph с конечным Ω итерация Φ_G от $\perp$ стабилизируется после конечного числа строгих добавлений, не превосходящего числа возможных node/atom memberships. После стабилизации ещё один, сотый или миллионный обход не добавляет отсутствующий atom.

Строки

100 лет
200 лет
300 лет
бесконечно

не являются introductions. Если graph, local introductions и policy неизменны, повторение того же оператора сохраняет тот же least fixed point.

Но из этого не следует утверждение обо всём неопisanном будущем. Новый sensor, goal, rule, узел, oracle-answer или policy образует новый input и новый contract-instance; новый graph возникает именно при изменении его V или E . Phase 3 устанавливает инвариант объявленного contract, а не запрет на появление любой последующей записи.

Также не утверждается, что произвольная бесконечная или non- ω -continuous semantics покрыта конечным checker. Limit-rule, transfinite stage или бесконечный read channel должны быть объявлены отдельно; их имя не является их реализацией.

18. Сильнейшие возражения

18.1. Рекурсия действительно создаёт новые результаты

Да. Новые payload и derived codes могут появляться по объявленным правилам из seeds. CP-LFP-3 относится к atoms, допущенным через Intro, и сохраняет для них origin/path; multi-premise derivation и его seed tree относятся к SG-LFP-3.

18.2. В цикле есть local introduction

Тогда SCC не source-free. Atom законно входит в least provenance и распространяется. Это положительный control, а не опровержение.

18.3. Greatest fixed point содержит atom без seed

Возможно по coinductive policy. Для $X \in \text{Fix}(\Phi_G) \dot{\vdash} \mu \Phi_G \dot{\vdash}$ каждый origin-tagged member из $X \dot{\vdash} \Phi_G$ не имеет finite introduction-path witness; для $X \in \text{Fix}(T_S) \dot{\vdash} \mu T_S \dot{\vdash}$ каждый code из $X \dot{\vdash} T_S$ не имеет finite seed-support tree. Это разные утверждения для unary provenance и support-hypergraph. Если nonleast state выбрана как accepted, selection-policy входит в полный input. Ошибкой было бы молча назвать её результатом CP-LFP-3 или SG-LFP-3.

18.4. Fixed-point semantics стандартна и не является каналом

Стандартность — ещё одна строка классификации. Два контракта с одинаковыми equations и разными policies могут выбрать разные states. Значит, policy различает outputs и входит в readset независимо от её распространённости.

18.5. Самопроверяющийся checker проверил себя

Можно принять exact trace самопроверки. Она получает status по checker-contract. Взаимная или рекурсивная структура trace не превращает этот status в независимое свидетельство правильности compiler, runtime, hardware, referent или мира. Соответствующие записи могут быть добавлены, но тогда они являются inputs.

18.6. Циклическая система действует на ракету

Действие и успех приняты в максимальной уступке. Они относятся к расширенной цепи formal, binding, implementation, sensor, event и criterion records. Ранний formula-code не становится всей цепью.

18.7. Constitutive convention действительно создаёт status

Да. Если convention объявляет определённый loop победой, proof-code или допустимым recursive object, formal status может быть детерминирован внутри практики. Convention является rule/policy input; introduction возникает только через отдельный protected Intro-constructor. Имя convention не предъявляет независимый referent.

18.8. В цикле невозможно найти первого предка

CP-LFP-3 не требует первого узла по круговому порядку. Она требует source introduction и конечный путь. SCC condensation локализует upstream или local source без искусственного разрыва цикла.

18.9. Negation или default rules меняют результат

Верно. CP-LFP-3 относится к объявленному monotone positive-union operator. Negation, default, defeat, time-dependent rule или nonmonotone update требуют отдельной semantics и включения policy в ρ . Применять theorem без этой посылки запрещено.

19. Точные фальсификаторы

19.1. Контрмодели теорем

CP-PATH-FALSIFIER — конечный exact object, удовлетворяющий CP admission gates и использующий действительное $P^\mu = \mu \Phi_G$, в котором выполняется хотя бы одно:

5. $(s, \tau, a) \in P^\mu_v$, но $a \notin \text{Intro}_\tau(s)$ или нет конечного пути $s \rightsquigarrow_G^* v$;
6. $a \in \text{Intro}_\tau(s)$ и $s \rightsquigarrow_G^* v$, но $(s, \tau, a) \notin P^\mu_v$.

SG-SUPPORT-FALSIFIER — конечные N, S, R , удовлетворяющие SG-domain и NONNULL, для которых существует $q \in N$ и выполняется хотя бы одно:

7. $q \in \mu T_S$, но конечного support tree с листьями в S нет;
8. предъявлено валидное конечное support tree для q с листьями в S , но $q \notin \mu T_S$.

FP-GROUNDING-FALSIFIER — fixed point $X \in \text{Fix}(T_S)$, который support-grounded, но $X \neq \mu T_S$, либо $X = \mu T_S$, но не support-grounded. Вторая ветвь одновременно дала бы SG-SUPPORT-FALSIFIER.

Это математические контрмодели; их определение не зависит от того, что какой-либо checker принял собственный output.

19.2. Дефекты executable conformance

Следующие случаи опровергают соответствие реализации опубликованному контракту, но сами по себе не являются контрмоделями теорем:

- checker принимает declared_fixed_point, отличный от untagged-проекции вычисленного $\mu \Phi_G$;
- checker извлекает protected atom из payload или принимает запрещённый kind/sort;
- checker принимает nonmonotone operator под CP-LFP-3;
- support checker принимает inference с одновременно пустыми premise_claim_ids и required_base_ids вместо status INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT и reason-code NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE;
- support checker принимает неверную claim-проекцию least closure, неверный support witness или неполную antichain минимальных bases.

Чтобы дефект реализации стал теоремным falsifier, требуется отдельно предъявить математический объект, сохраняющий все theorem premises и нарушающий соответствующую эквивалентность.

Не являются фальсификаторами:

- добавление local seed;

- добавление upstream introduction;
- изменение graph;
- выбор greatest/coinductive policy вместо least;
- скрытый oracle;
- изменение payload вместе с защищённым field;
- рассказ о референте, событии или причине вне declared relation.

Каждый такой случай меняет посылку или вход.

20. Исполняемая область

Phase 3 публикует два основных исполняемых слоя.

Cyclic provenance

cyclic_provenance_checker.py проверяет:

- конечность и точную структуру graph;
- protected introductions;
- инертность payload;
- итерацию positive-union operator снизу;
- равенство каждого declared_fixed_point untagged-проекции least fixed point;
- SCC condensation;
- один deterministic shortest-lex simple path witness для каждого достигающего introducer и exact minimum combined internal vertex cut при защищённых sources/root;
- отделение nonleast residue.

Основные статусы:

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

Контроли охватывают unseeded/seeded self-cycle, unseeded/seeded mutual cycle, nonleast residue, payload spoof и ациклический baseline.

Support grounding

support_grounding_checker.py проверяет:

- finite monotone inference hypergraph;
- seed-set и least closure;
- support trees;

- минимальные root bases;
- альтернативные и доминируемые bases;
- unseeded/seeded cycles;
- application/rocket support с goal и без goal;
- payload spoof.

Основные статусы:

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

Checker-output устанавливает только соответствие предоставленных JSON bytes объявленному алгоритму. Строка VALID не становится внешней истиной.

21. Самоприменение

Эта рукопись является exact record. Её заголовок, теоремные строки, формулы и слова полностью, источник, цикл не имеют исключения.

Статусы разделяются точными полями протокола. Для title и трёх theorem claims используются отдельные records, а не поле claim_ids:

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

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

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

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

Каждый text-record также содержит byte-exact поля exact_text, exact_text_sha256 и claim_sha256. Каждый artifact-record использует singular claim_id, kind: bounded_artifact, artifact, artifact_sha256, checker, expected_reason_codes, expected_observation, claim_sha256 и следующие literal status-поля:

```

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

```

Здесь `expected_artifact_status` — имя поля `self-audit manifest`; верхнеуровневый `runtime status` находится в соответствующем `checker verdict`. Wildcards `VALID-*`, `INVALID-*` и `INCOMPATIBLE-*` не являются `literal enum`-значениями `self-audit schema`.

Если рукопись ссылается на собственную доказанность, ссылка образует `edge/payload` данного `record`. Она не вводит `theorem-status`. Статус следует только из отдельно предъявленного `proof layer` и его `declared metatheory`; их написание также остаётся записью.

Если два допустимых `reading contracts` имеют одинаковый текст рукописи, но разные поля `applies_to_all_mathematics`, рукопись не выбирает это поле. Если новый `endorsement` добавлен позднее, он является новым `descendant record`.

22. Предшественники и статус новизны

Ни один внешний автор, школа, термин или смежная работа не является посылкой `CP-LFP-3`, `SG-LFP-3` или `FP-SEL-3`. Доказательства строятся из объектов, объявленных внутри пакета.

Строки `least fixed point`, `SCC`, `provenance` и `support` не дают исторический приоритет одним присутствием. В `Phase 3` внешний `collision/priority audit` не используется как основание и не завершён. `Schema-bound novelty record` содержит только `literal protocol`-поля:

```

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

```

`PREDECESSORS-AS-PREMISES: NONE` и `INTERNAL-PHASE-CONTRIBUTION: CP-LFP-3 + SG-LFP-3 + FP-SEL-3` являются `manuscript annotations`, а не несуществующими полями `phase3_self_claims.schema.json`. Их содержание не изменяет `novelty_status` и `priority_status`.

Позднее сравнительное исследование может изменить историческую маркировку. Оно не становится скрытой посылкой предъявленного доказательства.

23. Граница притязаний

Установлено

9. В `finite positive-union graph least provenance` равно множеству `origin-tagged introductions`, имеющих конечный путь к узлу.

10. Source-free SCC не порождает protected atom в least fixed point одной циркуляцией.
11. SCC с local/upstream protected introduction переносит atom, сохраняя origin tag.
12. SCC condensation сохраняет достижимость introductions и локализует local/upstream sources.
13. При protected-endpoints internal-cut contract atom исчезает из least root-provenance после удаления тогда и только тогда, когда удалённые internal nodes пересекают все пути от каждого достигающего introducer к root.
14. В finite positive support hypergraph после нормализации nullary constructors, с непустыми premises y ordinary rules, каждый least-closure root имеет конечное support tree с seed leaves.
15. Чистый self/mutual claim-cycle без seed и без альтернативного конечного seed-grounded support tree не образует grounded root в least closure.
16. Support-grounded fixed point оператора T_S единственна и равна μT_S ; nonleast fixed point содержит unsupported circular residue. Для unary Φ_G действует отдельное path-grounded следствие.
17. Fixed-point policy, меняющая выбранное состояние при одинаковых equations, является различающим input.
18. CP-AST переносит проекционную границу Phase 1 на least cyclic provenance.

Не установлено

- что все циклы бесполезны или недопустимы;
- что nonleast, greatest, stable или coinductive fixed points ложны;
- что все формальные теоремы неверны;
- что ни один знак никогда ни к чему не относится;
- что event-code является событием;
- что referent-code предъявляет референт;
- что cause-code является физической причиной;
- что goal-code оправдан;
- что meaning-code является смыслом;
- что declared graph/readset исчерпывает что-либо вне записи;
- что fixed-point policy правильна вне собственного контракта;
- что negation, default, defeat, probabilistic или temporal recursion покрыты positive-union theorem;
- что mutable или backdated record имеет засвидетельствованную физическую историю;
- что новый channel никогда не появится в будущем;
- что историческое происхождение математики из языка доказано;

- что Phase 3 имеет установленный мировой приоритет или глобальную новизну.

Эта граница не ослабляет theorem. Она запрещает циклу получить привилегию именно там, где origin отсутствует, и запрещает заголовку получить лишний квантор.

24. Итог

Замкнутая запись может быть вычислимо сложной. Рекурсивная программа может работать. Formal system может иметь self-reference. Два checker могут проверять друг друга. Controller и telemetry могут образовывать feedback loop. Fixed-point semantics может быть математически точной.

Но в least provenance protected atom не возникает потому, что линия вернулась к себе. Он присутствует там, куда существует путь от его introduction. Если introductions нет, пустой круг остаётся пустым. Если selected fixed point содержит больше, правило выбора является дополнительным входом. Если $q \in \mu T_S$, существует конечное seed-grounded support tree; принятие root по иной policy этим утверждением не охватывается.

Опровергнуто автономное циклическое притязание:

Замкнутая запись не становится первым источником отсутствующего protected atom только потому, что её элементы ссылаются друг на друга.

Не опровергаются условные рекурсивные выводы. Опровергается подмена переноса введённого atom его самозарождением.

*Цикл может не иметь первого узла.
У atom всё равно должен быть introduction.*

Статус источников

Работа продолжает внутреннюю авторскую линию beforeword и Phase 0-2 исследования математической границы записи. Внешние сравнительные источники в посылки не включены. Исторический приоритет и глобальная новизна в Phase 3 не оценивались.

CP-LFP-3 (CFP-3): теорема наименьшего происхождения для циклической записи

0. Статус обозначений

CP-LFP-3 — нормативный идентификатор теоремы. Короткий идентификатор CFP-3 используется как равнозначный alias. Расшифровка: **Cyclic Provenance — Least Fixed Point**.

Все объекты ниже являются точными элементами объявленного формального аппарата. Строки FORMAL, BINDING, EVENT, GOAL, REFERENT, CAUSE, APPLICATION и MEANING являются uppercase mathematical codes сортов; lowercase JSON-representatives фиксируются в § 2.2. Их наличие не предъявляет формулу, связь, событие, цель, референт, причину, применение или смысл вне записи.

Теорема не использует названное как посылку. Она использует только:

- равенство кодов;
- принадлежность конечным множествам;
- записанные направленные рёбра;
- защищённые конструкторы введения;
- наименьшую неподвижную точку явно заданного монотонного оператора.

payload не является источником типизированного происхождения. Совпадение строки в payload с atom-id, названием сорта, сериализацией введения или полным текстом этого файла ничего не добавляет в защищённое поле local_seeds.

1. Место относительно Phase 0–2

Phase 0 зафиксировала неустранимость зарегистрированной bridge-зависимости в append-only trace. Phase 1 установила проекционную границу: функция одинакового полного readset не выдаёт два разных обязательных target-code. Phase 2 локализовала различающий typed atom в ancestry конечного DAG.

CP-LFP-3 снимает только одно ограничение Phase 2: граф записи может содержать направленные циклы. Для этого рекурсивное вычисление provenance заменяется на явно выбранную наименьшую неподвижную точку. Остальные границы не ослабляются:

- происхождение остаётся origin-tagged;
- target фиксируется отдельно;
- полный readset входит в base-проекцию;
- typed record не становится своим референтом;

- ребро provenance не становится физической причиной.

На DAG результат CP-LFP-3 точно сводится к ancestor-union Phase 2.

2. Конечная циклическая запись

2.1. Граф

Пусть

$$G=(V,E)$$

— конечный ориентированный граф. Петли и направленные циклы разрешены. Ребро

$$u \rightarrow v$$

означает только записанное отношение u входит в объявленный dependency-readset узла v . Оно не означает физическую передачу, причинное действие, временное предшествование или семантическую зависимость.

У каждого узла имеются защищённые поля:

- node_id;
- kind;
- СПИСОК parents;
- local_seeds — исполняемое имя bucket-ов atom-id защищённых introductions; сами bucket-ы untagged, а origin tag добавляется парой с node_id;
- declared_fixed_point — исполняемая untagged-проекция заявленного least provenance для этого узла;
- произвольный payload.

Для CP-LFP-3 порядок birth_rank не требуется: в цикле строгий рост ранга вдоль каждого ребра невозможен. Физическое время из отсутствия ранга также не выводится.

2.2. Сорта и atom-id

Phase 3 использует расширенный конечный реестр защищённых сортов

$$T_3 = \{ \underset{\text{FORMAL}}{\text{FORMAL}}, \underset{\text{BINDING}}{\text{BINDING}}, \underset{\text{EVENT}}{\text{EVENT}}, \underset{\text{GOAL}}{\text{GOAL}},$$

Поле JSON sort представляет эти восемь mathematical codes фиксированной биекцией FORMAL ↔ "formal", BINDING ↔ "binding", EVENT ↔ "event", GOAL ↔ "goal", REFERENT ↔ "referent", CAUSE ↔ "cause", APPLICATION ↔ "application", MEANING ↔ "meaning". Отдельное поле узла kind использует uppercase literal values и допускает также RELAY и VERDICT.

Path-теорема параметрична по конечному реестру сортов. Опубликованный executable contract Phase 3 перечисляет ровно восемь lowercase JSON-representatives элементов T_3 ; его машинный статус относится только к предоставленному finite JSON record и не переносится на иной sort по одному совпадению имени.

Первые четыре сорта сохраняют роли Phase 2. Дополнительные сорта также являются только сортами записей. В частности:

- REFERENT — запись referent-code, не референт;
- CAUSE — запись cause-code относительно объявленного контракта, не физическая причина;
- APPLICATION — запись application-code, не факт применения;
- MEANING — запись meaning-code, не смысл;
- GOAL — запись goal-code, не оправданность и не принятие цели;
- EVENT — запись event-code, не событие.

Для каждого сорта τ задан конечный реестр atom-id A_τ . Для каждого узла v :

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

— конечное множество atom-id, введённых защищённым конструктором этого сорта.

Совместимый контракт обязан отдельно проверять допустимые пары node kind / introduced sort. Произвольный узел не получает права вводить любой сорт из-за текста своего payload.

2.3. Origin tags

Локальное origin-tagged введение:

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

Общий конечный реестр возможных origin-tagged атомов:

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

Тройки (v, τ, a) и (w, τ, a) различны при $v \neq w$. Строка, изображающая такую тройку в payload, не является защищённым origin tag.

Checker, который распространяет только untagged пары (τ, a) , проверяет untagged-проекцию path-теоремы. Для статуса полного origin-tagged provenance он обязан либо переносить triples (s, τ, a) на каждом узле, либо исчерпывающе реконструировать и предъявлять все introducer-node ids для каждого принятого atom во всех проверяемых узлах. Один origin-witness для одного query не равен полной origin-tagged closure.

3. Оператор циклического происхождения

Рассматривается конечная полная решётка

$$L = (2^{\Omega})^V$$

с покоординатным включением. Элемент $X \in L$ назначает каждому узлу множество origin-tagged атомов $X(v)$.

Оператор CP-LFP-3:

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

Он монотонен:

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

Нормативное provenance определяется не произвольным решением уравнения $X = \Phi_G(X)$, а итерацией от дна:

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

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

Поскольку V конечно и новые атомы только объединяются, итерация стабилизируется. Переход $P^0 \rightarrow P^1$ помещает local introductions; затем достаточно не более $|V| - 1$ меняющих состояние propagation-переходов, потому что вся достижимость имеет простой путь длины не более $|V| - 1$. Следовательно, $P^{|V|} = \mu \Phi_G$, хотя реализации может потребоваться ещё одна проверка равенства состояний, чтобы обнаружить стабилизацию.

Checker обязан вычислять именно эту least closure. Исполняемая реализация инициализирует current untagged-проекцией

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

то есть пропускает детерминированное хранение P^0 и перехода $P^0 \rightarrow P^1$. Поэтому iteration_count считает только последующие переходы, которые строго изменили хотя бы один untagged bucket; он не считает финальную проверку без изменения. Это runtime-счётчик, а не индекс n в доказательстве, и его смещение не меняет $\mu \Phi_G$. Приём любого другого fixed point как declared_fixed_point несовместим с CP-LFP-3.

4. CP-LFP-3: теорема пути

Обозначение

$$s \sim_G^* v$$

означает существование конечного направленного пути из s в v , включая путь длины ноль при $s = v$.

Теорема квантифицирует существование пути и не выбирает его сериализацию. Executable verdict отдельно выдаёт в `query_results[].path_witnesses[].node_ids` один `deterministic shortest-lex simple path`: сначала минимальна длина, затем лексикографически сравнивается последовательность `node_id`. Эта witness-policy не является дополнительной посылкой path-эквивалентности.

Теорема CP-LFP-3

Для каждого $v \in V$:

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

То есть наименьшее циклическое provenance содержит ровно защищённые origin-tagged введения, достигающие узел по конечным записанным путям.

Доказательство

Для $n \geq 1$ вводится

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

Равенство $P^n(v) = R_n(v)$ устанавливается индукцией.

При $n = 1$:

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

поскольку единственные пути длины меньше одного — рефлексивные пути длины ноль.

При условии $P^n(u) = R_n(u)$ для всех u :

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

Первое слагаемое даёт пути длины ноль. Второе даёт ровно те пути в v , которые заканчиваются ребром $u \rightarrow v$, а до u имеют длину меньше n . Поэтому оно даёт пути положительной длины меньше $n + 1$. Следовательно,

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

Объединение по всем n даёт все и только конечные пути. В конечном графе каждый достижимый узел имеет простой путь, поэтому стабилизация конечна. $\square$

Жёсткое следствие

Если origin-tagged atom входит в $OProv_G^\mu(v)$, то существует точный узел s , где этот atom введён защищённым конструктором, и конечный путь $s \rightsquigarrow_G^* v$.

Обратная строка также точна: всякое защищённое введение, имеющее путь к v , входит в least provenance v .

Ни одна часть теоремы не устанавливает, что atom-id предъявляет названное.

5. Цикл не является источником

5.1. Точное условие отсутствия источника

Для $C \subseteq V$ входящий конус определяется как

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

Для фиксированного atom-code (τ, a) условие NO-UPSTREAM-INTRO означает

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

Требуется отсутствие введения во всём входящем конусе, а не только внутри C . Цикл без локального введения может законно перенести atom, пришедший по входящему пути.

Следствие CP-NOSOURCE

Если NO-UPSTREAM-INTRO выполнено, то для каждого $v \in C$:

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

Это немедленно следует из CP-LFP-3.

Точная публичная строка:

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

5.2. Положительный контроль

Пусть $v_0 \rightarrow v_1 \rightarrow v_0$, а внешний для компоненты узел s вводит (τ, a) и имеет ребро $s \rightarrow v_0$. Тогда atom входит в least provenance обоих узлов цикла. Это не контрпример: источник — s , а цикл только распространяет введение.

Если atom локально вводит v_0 , он также достигает всей своей strongly connected component. Локальный introducer является источником; сам факт замыкания пути источником не является.

6. SCC-конденсация

Пусть $SCC(G)$ — разбиение V на strongly connected components. Конденсация

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

является DAG. Для компоненты C задаётся

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

Теорема CP-SCC

Для узла v в компоненте C_v :

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

Доказательство

Направленный путь между узлами индуцирует путь между их SCC в конденсации. Обратно, путь компонент можно поднять до пути между нужными узлами, а внутри каждой SCC все узлы взаимно достижимы. Поэтому source-node достигает v в G тогда и только тогда, когда его компонента достигает C_v в D_G . Подстановка этого равенства достижимости в CP-LFP-3 даёт формулу. $\square$

Конденсация удаляет повторное хождение по циклу, но сохраняет каждое origin-tagged введение. Она показывает точную функцию SCC: распространять входящие и локальные введения по своей области достижимости.

Восстановление Phase 2

Если G — DAG, каждая SCC одноэлементна. Тогда CP-SCC становится

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

то есть IADC/ancestor-union Phase 2.

7. Разрез в циклическом графе

Обозначим

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

Пусть $r \in V$ — защищённый root. Для fixed untagged code (τ, a) множество достигающих root introducers:

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

Обозначим через $P_G(s, r)$ множество конечных направленных путей из s в r , а через $V(p)$ — множество вершин пути p .

В internal-cut contract все introducers из $S_{\tau,a}^r$ и root защищены. Пусть

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

Операция блокировки удаляет internal nodes B и инцидентные рёбра, не выполняет rewiring и затем пересчитывает least fixed point на $G - B$.

Теорема CP-CUT

После блокировки untagged atom (τ, a) отсутствует в provenance r тогда и только тогда, когда B пересекает каждый направленный путь от каждого $s \in S_{\tau,a}^r$ к r :

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

Доказательство

По CP-LFP-3 atom остаётся у root после блокировки ровно тогда, когда существует surviving path от защищённого introducer к r . Отрицание существования такого пути означает, что B пересекает каждый introducer-to-root path. Наличие циклов не меняет критерий: любой walk, достигающий root, содержит простой path с теми же концами. $\square$

Если $S_{\tau,a}^r = \emptyset$, nested cut_status равен ABSENT, а rank равен null: пустой hitting set не сериализуется как cut-certificate. Если хотя бы один introducer совпадает с root или имеет прямое ребро в root, internal cut отсутствует: защищённые endpoints нельзя удалить, и cut_status равен UNBLOCKABLE-PROTECTED-ENDPOINT. Во всех остальных nonempty случаях минимальный по мощности B является finite combined internal vertex-cut certificate со status CUT-CERTIFIED. Его размер измеряет только структурную избыточность записанных маршрутов происхождения. Он не измеряет истинность, физическую причинную силу, полезность или смысл atom-code.

8. Композиция с AST-1

Пусть $K \neq \emptyset$ — объявленный класс exact-record индексов. Base-проекция

$$\rho: K \rightarrow R$$

обязана включать либо однозначно фиксировать:

- полный G , node kinds и root v ;
- реестры сортов и atom-id;
- структуру, реестры и типовую политику introductions; exact значения introductions, образующие root provenance, вынесены в η_v^μ и не дублируются в ρ , если evaluator не читает их иным способом;
- оператор Φ_G и правило выбора least;
- evaluator contract;
- всякий seed, transcript, schedule, oracle-output или другой фактически прочитанный input.

Если какой-либо прочитанный компонент различается, ρ также должна различаться.

Cyclic provenance view определяется как

$$\eta_v^\mu(k) = \text{canon}\left(O\text{Prov}_{G_x}^\mu(v)\right),$$

где canon инъективно на точных конечных множествах origin-tagged triples.

Пусть $Q: K \rightarrow Y$ — независимо фиксированная target-coordinate, а

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

для тотальной функции F . Если на всём K :

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

то

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

Доказательство

Если $\rho(k_0) = \rho(k_1)$ и $\eta_v^\mu(k_0) = \eta_v^\mu(k_1)$, функция F получает одинаковый аргумент и выдаёт одинаковый output-code. Равенство $C_v = Q$ переносит это равенство на Q . $\square$

Origin witness при циклах

Если $\rho(k_0) = \rho(k_1)$, но $Q(k_0) \neq Q(k_1)$, то при условии корректного экспорта

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

Инъективность canon даёт origin-tagged triple в симметрической разности. По CP-LFP-3 triple имеет защищённый introducer и конечный introducer-to-root path в том случае, где он присутствует. Путь может проходить через SCC; SCC не заменяет introducer.

Это распространяет структурную локализацию AST-1 на cyclic records. Оно не превращает target-code $Q(k)$ в референт, событие, причину, применение, цель или смысл.

9. Полное замыкание не создаёт внешний target

В view (ρ, η_v^μ) можно включить:

- весь cyclic graph;
- все итерации от $\perp$;
- конечный least fixed point;
- все formal payload;
- proof traces;
- self-reference strings;
- controller states и feedback transcripts, если они записаны.

Если два допустимых индекса имеют одинаковый полный view, но независимо заданные acceptable target-sets $A(k_0), A(k_1) \subseteq Y$ имеют пустое пересечение, то zero-error selector из этого view отсутствует:

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

влечёт

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

Это прямое применение RBI-1/FDT-2 после вычисления циклического closure. Цикл не делит fiber, если полный прочитанный cyclic view одинаков.

Для каждого exact sort FORMAL, BINDING, EVENT, GOAL, REFERENT, CAUSE, APPLICATION, MEANING условие проверяется отдельно. purpose и justification здесь могут быть только отдельно типизированными claim-role codes, а не неявными дополнительными сортами. Успешная selectability одного target не наследуется другим target.

10. Ковариантность переименования

Пусть π — полная биекция, которая согласованно переименовывает:

- node-id;
- sort-id;
- atom-id;

- graph edges;
- защищённые introductions;
- все соответствующие поля оператора и evaluator contract.

Частичное переименование payload без транспортировки правил сюда не относится.

Для транспортированного графа πG :

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

Доказательство

Биекция сохраняет и отражает introductions и конечную достижимость. Применение CP-LFP-3 к обеим сторонам даёт одинаковые транспортированные triples. Эквивалентно, $\pi \circ \Phi_G = \Phi_{\pi G} \circ \pi$, а изоморфизм решёток переносит least fixed point. ▢

Поэтому форма глифа или имя event не имеют привилегии внутри теоремы. Если вывод меняется при полном структурно согласованном recoding, нарушена предпосылка полного транспорта. Если меняется только внешний assignment при неизменном formal view, нужен отдельный binding-channel.

11. Снимки и точная граница неретроактивности

Phase 2 запрещала новые рёбра к старым узлам через forward birth-rank. Циклический контракт такого запрета автоматически не содержит.

Пусть (G, I) — старый snapshot, (G', I') — расширенный snapshot, сохраняющий старые узлы, рёбра и introductions, но допускающий новые рёбра. Для старого root r :

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

состоит ровно из тех introductions, для которых в G' появился путь к r , отсутствовавший в G , включая introductions новых узлов.

Следовательно, старое provenance остаётся неизменным тогда и только тогда, когда расширение не создаёт ни одного нового introducer-to-root path. Добавление обратного ребра может изменить closure старого root в новом snapshot; это не переписывает байты старого snapshot, но снимает безусловную фразу о неретроактивности нового графа.

Совместимые варианты фиксации:

- 19.immutable versioned snapshot: старый root вычисляется только в старом G ;
- 20.запрет новых путей к защищённым старым roots;

21.новый root/version для нового closure.

Строка времени или hash не является физическим временем сама по себе.

12. Admission gates

Instance получает статус CP-LFP-3 только если выполнены все условия:

- 22. V, E, T_3, A_t и introductions конечны и структурно валидны;
- 23.каждый parent-id разрешается;
- 24.operator точно равен (CP) либо заранее объявленному эквивалентному union-operator;
- 25.operator монотонен на конечном powerset-domain;
- 26.provenance вычислено итерацией от $\perp$, а не выбрано как произвольный fixed point;
- 27.каждый untagged bucket declared_fixed_point точно равен untagged-проекции вычисленного $OProv_G^\mu$ в соответствующем узле;
- 28.origin tag связан с реальным node_id и защищённым introduction соответствующего сорта;
- 29.payload не читается как introduction;
- 30.полный evaluator-readset включён в ρ ;
- 31.target и acceptable sets фиксированы независимо от output.

Немонотонная рекурсия, negation-as-failure и удаление атомов получают OUTSIDE-CP-LFP-3-NONMONOTONE. Скрытые clock, scheduler, oracle или mutable state снимают полноту readset; бесконечный instance находится вне конечного исполняемого контракта.

Общая set-theoretic path-формула допускает и бесконечный граф для union-operator, но исполняемый Phase 3 contract ограничен конечными records. Машинный статус не переносится на бесконечный объект.

13. Нормативные статусы

Literal верхнеуровневые schema-statuses checker: для instance-verdict — VALID-CYCLIC-PROVENANCE-LFP, INVALID-CYCLIC-PROVENANCE, INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT; для contract-audit verdict — CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS, CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL. ABSENT, CUT-CERTIFIED и UNBLOCKABLE-PROTECTED-ENDPOINT являются значениями вложенного поля query_results[].minimum_internal_vertex_cut.cut_status, а не верхнеуровневого status. Остальные строки таблицы — protocol-level labels теорем и controls.

Статус	Точное условие	Чего статус не устанавливает
VALID-CYCLIC-PROVENANCE-LFP	каждый <code>declared_fixed_point</code> равен <code>untagged-проекции $OProv_G^\mu$</code>	истинность payload или target
CYCLE-PROPAGATES-INTRODUCED	atom имеет upstream/local introducer и path через SCC	что atom-id предъявляет названное
NO-ORIGINLESS-ATOM-IN-LFP	для каждого atom в least provenance найден introducer и finite path	отсутствие события, цели или смысла вне записи
SCC-CONDENSATION-PRESERVES-PROVENANCE	CP-SCC выполнена на exact instance	физическую эквивалентность систем
ABSENT	достигающих root introducers для query atom нет	отсутствие названного вне записи
CUT-CERTIFIED	exact minimum combined internal cut разъединяет все protected sources и root	физическую причинность cut-узлов
UNBLOCKABLE-PROTECTED-ENDPOINT	local/direct protected path не имеет internal cut	неустранимость atom вне этого contract
CYCLIC-AST-SEPARATED	target-conflict сопровождается различием η_v^μ	правильность target assignment вне контракта
PAYLOAD-SPOOF-REJECTED	строка не принята как protected introduction	ложность строки
OUTSIDE-CP-LFP-3-NONMONOTONE	operator немонотонен	отрицательный результат о другом operator

Названия статусов остаются строками протокола.

14. Обязательные контроли

32. **Пустая self-loop.** $v \rightarrow v$, introductions пусты. Least provenance пусто.

33. **Два взаимных узла без источника.** $u \leftrightarrow v$, introductions пусты. Least provenance обоих пусто.

34. **Входящий source.** s вводит atom и направлен в $u \leftrightarrow v$. Atom достигает обоих узлов цикла.

35. **Локальный source.** Один узел SCC вводит atom. Atom достигает всей SCC и каждого downstream node.

36. **Payload spoof.** Self-loop содержит строку `EVENT: flight occurred`, но `local_seeds.event=[]`. Event provenance пусто.

37. **Origin spoof.** `declared_fixed_point.event` содержит e в узле без reachable `local_seeds.event` introducer. Instance отвергается.

38.**SCC collapse.** Provenance каждого root совпадает до и после конденсации с сохранением origin tags.

39.**Cyclic cut.** При двух маршрутах через цикл неполный cut сохраняет atom; hitting set всех simple paths устраняет.

40.**Back-edge update.** Новый introducer получает путь к старому root: новый snapshot получает atom, старый versioned snapshot не меняется.

41.**Complete recoding.** Полное биективное переименование коммутрует с least closure.

42.**Partial recoding.** Переименование carrier без parser/rule transport не выдаётся за CP-RECODE.

43.**Completion conflict.** Полный cyclic view одинаков, singleton target-codes различны: автономный selector отсутствует.

44.**Overlap control.** Acceptable sets различаются, но имеют общий code: общий zero-error code допускается.

45.**Constant target.** Target постоянен на view-fiber: факторизация допускается.

Положительные контроли 3, 4, 13 и 14 обязательны: теорема не должна быть реализована как безусловный отказ принимать cycle, propagation или selector.

15. Точные фальсификаторы

Следующие falsifiers являются математическими контрмоделями при сохранении admission gates. Само принятие ошибочного JSON checker-ом ещё не является такой контрмоделью.

CP-PATH-FALSIFIER

CP-LFP-3 опровергнута на exact instance, если при сохранении всех admission gates предъявлено хотя бы одно:

46.origin-tagged atom входит в вычисленное $OProv_G^\mu(v)$, но не существует соответствующего защищённого introducer и finite path introducer-to-v;

47.защищённый introducer имеет finite path к v , но соответствующий origin-tagged atom отсутствует в $OProv_G^\mu(v)$.

CP-SCC-FALSIFIER

Конденсация сохраняет exact introductions и reachability между компонентами, но formula CP-SCC даёт другое root provenance.

CP-CUT-FALSIFIER

При удалении internal nodes без rewiring, защищённых introducers/root и повторном least closure:

- все introducer-to-root paths пересечены, но atom остаётся; либо
- surviving introducer-to-root path существует, но atom исчезает.

CP-AST-FALSIFIER

Существуют k_0, k_1 с одинаковыми ρ и η_v^μ , разными Q , причём тотальная функция F действительно получает только эти два аргумента и удовлетворяет $F(\rho, \eta) = Q$ на всём объявленном K .

Скрытый oracle, разный scheduler, неполная ρ , изменённый graph, другой fixed-point policy, post hoc target или semantic assertion без record-code снимают предпосылку; они не являются фальсификаторами.

Отдельно: defect executable conformance

Если checker принимает declared_fixed_point, отличный от untagged-проекции вычисленного $OProv_G^\mu$, извлекает introduction из payload, принимает запрещённую kind/sort пару, выдаёт неверный origin/path witness либо смешивает nested cut-status с верхнеуровневым status, это дефект executable conformance. Он фальсифицирует заявление о соответствии реализации контракту; CP-LFP-3 он опровергнет только после предъявления отдельного математического counterexample к path-эквивалентности.

16. Самоприменение и максимальная строка

Этот файл сам является carrier. Наличие в нём строки CP-LFP-3 не создаёт теорему вне выбранного чтения; наличие строки MEANING не создаёт смысл; наличие доказательства не предъявляет внешний референт. Внутри объявленной метатеории записанная последовательность выводит условную path-формулу. Именно эта граница применяется и к самому документу.

Максимальная точная формулировка Phase 3:

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

Не заявляется:

- что все формальные выводы ложны;
- что циклы физически ничего не делают;
- что referent/event/cause/application/purpose/meaning отсутствуют;
- что declared graph является физической причинной системой;
- что least fixed point является единственно допустимой семантикой для любого другого контракта;
- что код статуса удостоверяет названное им.

SG-LFP-3 (GST-3): теорема опоры и исходных введений

0. Нормативный статус

SG-LFP-3 — нормативный идентификатор. GST-3 — короткий alias: **Grounded Support Theorem**. Файл расширяет unary propagation CP-LFP-3 до конечных положительных AND/OR-правил.

Здесь слова `seed`, `support`, `rule`, `root`, `proof` и `grounded` обозначают поля и отношения exact-record контракта. Они не удостоверяют основание вне записи, истинность, наблюдение, причину, цель, референт или смысл. В частности:

- `seed` — защищённое нульарное введение `record-code`;
- `support tree` — конечное дерево `rule-id` и `code-id`;
- `grounded` — наличие такого дерева;
- `root` — выбранный `code-id` запроса.

Наличие строки «это основание» в `payload` не создаёт `seed`. Наличие строки «из A следует B» не создаёт `hyperrule`. Оба объекта регистрируются защищёнными конструкторами.

1. Зачем нужен AND/OR-слой

CP-LFP-3 рассматривает перенос одного и того же `origin-tagged atom` по рёбрам. Формальная запись часто имеет более общий вид:

- один `output` требует одновременно несколько `premise-code` — AND;
- один `output` может быть получен разными `rule-id` — OR;
- `rule dependencies` могут образовывать `cycles`;
- один `root` может иметь несколько несовместимых или перекрывающихся `seed-базисов`.

SG-LFP-3 отвечает на точный вопрос:

Когда `root-code` входит в наименьшее замыкание конечного положительного `rule-record` и какой конечный набор защищённых `seeds` поддерживает это вхождение?

Ответ не использует семантическое чтение `code-id`. Он использует только `membership`, `subset` и конечную итерацию.

2. Конечный support-contract

2.1. Абстрактный code universe

Пусть N — конечное непустое множество exact `code-id`. Элементы N могут иметь типизированную форму

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

где v — origin node, τ — sort-id, a — atom-id. Теорема не требует такого разложения и применима к любым точным code-id.

Пусть

$$S \subseteq N$$

— множество защищённых seed/base codes. В абстрактной теореме seed может одновременно быть code, участвующим в rules. В executable refinement ниже namespaces base и claim разделены tagged-вложением. Одинаковый payload при другом защищённом id не склеивает records.

2.2. Положительные hyperrules

Конечное множество правил R состоит из записей

$$r = (id_r, Prem(r), Head(r)),$$

где

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

Все premise-code одного правила требуются одновременно. Разные rules с одним Head являются альтернативами.

Нормативное условие области после нормализации

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

означает

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

Нулевой constructor $\emptyset \rightarrow h$ не входит в $R_{ordinary}$ молча. До применения теоремы он нормализуется в fresh protected base $b_r \in S$ и ordinary rule $\{b_r\} \rightarrow h$. Один допустимый convention-id для b_r :

RULE-SEED:<rule-id>

2.3. Точное executable-вложение

Пусть

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

Теги делают $B \cap C = \emptyset$, даже если одинаковая raw-строка разрешена в двух namespaces. Executable record вкладывается в абстрактную систему с $N = B \sqcup C$, полным seed-set $S = B$, а каждый inference r задаёт

$$id_r = inference_{id}, Prem(r) = \{(BASE, b) : b \in required_{id}\} \cup \{(CLAIM, c) : c \in premise_{id}\},$$

$$Head(r) = (CLAIM, conclusion_{id}).$$

Такое вложение сохраняет AND всех base/claim premises и claim-проекцию least closure. Полный executable seed-set равен $S = B$. При проверке candidate basis используется новое обозначение $B' \subseteq B$: в операторе seed-аргумент S заменяется на B' , тогда как N и R остаются неизменными. Нормализованный nullary constructor создаёт protected base и inference к нужному conclusion_id, явно перечисляющий этот base_id в required_base_ids. Checker сам не нормализует input и не требует prefix RULE-SEED; его точное executable domain-condition — для каждого inference непусто объединение premise_claim_ids и required_base_ids. Если оба массива пусты, он возвращает status INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT и reason_codes=["NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE"].

2.4. Точное соответствие runtime fields

Runtime field	Абстрактный объект или точный эффект
bases[].base_id = b	tagged code $(BASE, b) \in B = S$
bases[].role_code	проверяемая metadata-классификация; копируется в base_catalog, но не входит в premise-test
claims[].claim_id = c	tagged code $(CLAIM, c) \in C$
inferences[].inference_id	id_r
inferences[].conclusion_id	raw representative head $(CLAIM, c)$
inferences[].premise_claim_ids	claim-часть $Prem(r)$
inferences[].required_base_ids	base-часть $Prem(r)$
claims[].declared_minimal_supports	объявленная exact antichain минимальных subsets raw base_id; checker требует её fixed-point equality и равенство вычисленному least frontier
query.claim_ids	непустая проекция уже вычисленного результата; не меняет closure
claim_grounding[].grounded	true ровно при непустом вычисленном frontier данного claim

Runtime field	Абстрактный объект или точный эффект
claim_grounding[].minimal_supports	вычисленная inclusion-minimal antichain raw base_id
claim_grounding[].producer_inference_ids	все объявленные inference с данным conclusion_id, а не только inference выбранного witness
support_tree_witnesses	по одному детерминированному witness на каждую пару claim/minimal support
ungrounded_claim_ids	claims с пустым least frontier
unseeded_cycle_claim_sets	циклические claim-SCC, все claims которых имеют пустой least frontier
iteration_count	число строгих переходов frontier-итерации до стабилизации
artifact_sha256	canonical digest всего принятого input record

2.5. Инертность payload и canonical digest

Runtime проверяет payload каждого base, claim и inference как допустимое JSON-значение с `MAX_PAYLOAD_DEPTH = 32` (при общем `bound record-depth 64`), но функция `infer_once` не читает payload как premise, base, rule или head. Rule-support определяется только защищёнными ids и contract policy; похожая строка внутри payload не извлекается как правило.

Для принятого record поле `artifact_sha256` равно `"sha256:" + hex(SHA256(canonical_bytes(value)))` для полного input value, включая contract, query и все payload values. В профиле `bw-cjson-2-python-reference` `canonical_bytes` — UTF-8 JSON с отсортированными object keys, separators `,` и `:` без добавочного whitespace, `ensure_ascii=false` и запретом non-finite numbers. Это digest канонического значения, а не исходных whitespace или порядка object keys; Unicode не нормализуется. Поэтому изменение канонического payload value может изменить `artifact_sha256`, но при неизменных защищённых полях не меняет `N`, `S`, `R`, least frontier и support witnesses. Наличие в payload полного JSON правила, base, дерева или результата само по себе support не создаёт.

3. Наименьшее AND/OR-замыкание

На конечной решётке 2^N задаётся оператор

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

Он монотонен:

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

Итерация от дна:

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

определяет

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

Поскольку N конечно и operator только добавляет коды, итерация стабилизируется за конечное число раундов.

cl означает только membership в least record-closure. Строка

$$q \in Cl_R(S)$$

не означает, что payload q истинен или что названное им существует.

4. Конечное support tree

4.1. Определение

Support tree для root-code $q \in N$ — конечное корневое дерево D , у которого каждый tree-node помечен code-id из N , и выполнены условия:

- 48.root помечен q ;
- 49.каждый лист помечен element $s \in S$;
- 50.каждый внутренний узел с label h содержит exact rule-id id_r , где $Head(r) = h$;
- 51.множество дочерних labels точно равно $Prem(r)$, каждая premise представлена ровно одним child, а сериализация children использует канонический порядок контракта;
- 52.каждый rule-id и code-id разрешается в предъявленном record.

В абстрактном дереве каждый seed-leaf помечен некоторым code $s \in S$. В executable embedding полный seed-set равен $S = B$, поэтому s имеет tagged-форму $(BASE, b)$, а raw строка $base_id = b$ является только его compact representative после разрешения в таблице bases. Compact verdict не создаёт для такого leaf отдельный JSON tree-node: b записывается в `required_base_ids` родительского claim-узла, а множество всех base-leaves данного subtree — в `support_base_ids`. Поле `premise_trees` содержит только claim-children. В executable embedding abstract rule-id представлен полем `inference_id`, а head — tagged `conclusion_id`. Следовательно, отсутствие отдельного JSON child-node для base не удаляет abstract leaf и не отождествляет raw строку с элементом S без namespace-tag.

Один и тот же code-id может встречаться в разных ветвях. Дерево обязано быть конечным. Бесконечная ветвь, заканчивающаяся ссылкой на саму себя, не является support tree.

Множество abstract seed-codes в листьях:

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

Повторяющийся seed учитывается один раз в set-valued basis; если конкретному контракту важна кратность ресурса, требуется отдельная multiset-семантика, не SG-LFP-3.

4.2. Высота

Высота leaf равна нулю. Высота внутреннего узла равна единице плюс максимум высот children. Конечность высоты является операционным свидетелем отсутствия бесконечного регресса внутри этого дерева.

5. SG-LFP-3: теорема support tree

Теорема SG-LFP-3

Для каждого $q \in N$ эквивалентны:

$$q \in \mu T_S;$$

существует конечное support tree для q с листьями в S .

В boxed-форме:

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

Доказательство $(A) \Rightarrow (B)$

Пусть n — первый номер, для которого $q \in C_S^{n+1}$.

Если $q \in S$, одноузловое дерево с leaf q является support tree. Иначе существует rule r с $Head(r) = q$ и

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

Для каждой premise $p \in Prem(r)$ номер её первого появления строго меньше номера первого появления q . Индукция по first-entry rank даёт конечное support tree для каждого p . Новый root q получает rule-id r , а эти деревья присоединяются как children. Получено конечное support tree для q . $\square$

Доказательство $(B) \Rightarrow (A)$

Индукция по высоте support tree. Каждый leaf принадлежит $S \subseteq T_S(X)$ для любого X .

Пусть все children внутреннего узла h входят в closure. Их конечное множество является $Prem(r)$ объявленного rule r с head h . На некоторой общей конечной итерации все premises уже присутствуют; следующая итерация добавляет h . Следовательно, root входит в μT_S . $\square$

Seed-ancestry следствие

Каждое вхождение root в least closure имеет хотя бы один конечный путь поддержки к seed-code $s \in S$. Циклическая ссылка может находиться внутри выбранного record, но ни одна ветвь принятого support tree не заканчивается циклом: в абстрактной теореме она заканчивается $s \in S$, а в executable embedding этот leaf представлен exact `base_id`.

Это структурная seed-ancestry, не историческая, физическая или смысловая ancestry.

6. Минимальные root bases

6.1. Basis

Кандидат $B' \subseteq S$ называется root basis для q , если

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

где правила R неизменны, а доступные seeds ограничены точным множеством B' .

Basis минимален по включению, если

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

и для каждого собственного $B'' \subsetneq B'$:

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

Семейство всех inclusion-minimal bases обозначается как

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

Если $q \notin Cl_R(S)$, это семейство пусто.

Теорема SG-BASIS

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

Доказательство

Каждое support tree с leaf-set L по SG-LFP-3 показывает

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

поэтому L — basis.

Если inclusion-minimal leaf-set L не был minimal basis, существовал бы proper sub-basis $B \subsetneq L$. SG-LFP-3 для seed-set B дало бы support tree с leaf-set $L' \subseteq B \subsetneq L$. Это противоречит минимальности

L среди leaf-sets support trees. Следовательно, каждый inclusion-minimal leaf-set является minimal basis.

Обратно, пусть B — minimal basis. SG-LFP-3 для seed-set B даёт support tree D с $Leaves(D) = L \subseteq B$. Если $L \subsetneq B$, то $q \in Cl_R(L)$, что противоречит минимальности B . Поэтому $L = B$. Минимальность basis совпадает с inclusion-minimal leaf-set. $\square$

6.2. May- и must-seeds

Если $B_{min}(q) \neq \emptyset$, задаются:

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

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

MaySeed перечисляет seeds, встречающиеся хотя бы в одном минимальном basis; MustSeed — seeds, встречающиеся в каждом минимальном basis. Пустой MustSeed не означает отсутствие поддержки: root может иметь несколько альтернативных непересекающихся bases.

Эти множества не измеряют истинность, важность или причинную необходимость вне support-contract.

6.3. Минимальный размер

Если $B_{min}(q) \neq \emptyset$, задаётся

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

При пустом семействе $\beta(q)$ не определено. В остальных случаях это минимальная мощность seed-basis в данной записи, а не количество физических причин, степень уверенности или величина смысла.

7. Точное блокирование root

Пусть $D \subseteq S$ — seeds, отключённые до повторного least closure. Root отсутствует после блокировки ровно тогда, когда ни один minimal basis не сохранился целиком.

Теорема SG-HIT

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

Доказательство

Если некоторый $B \in B_{min}(q)$ не пересекает D , то $B \subseteq S \setminus \dot{I}$, и монотонность даёт root после блокировки.

Обратно, если root остаётся после блокировки, SG-LFP-3 даёт конечное support tree с leaf-set $L \subseteq S \setminus \dot{I}$. В конечном L существует inclusion-minimal sub-basis $B \subseteq L$. Тогда $B \in B_{min}(q)$ и $B \cap D = \emptyset$, противоречие условию hitting. $\square$

Minimum hitting set семейства $B_{min}(q)$ является seed-blocking certificate. Это не физический intervention и не доказательство внешней причинности.

8. Циклы в AND/OR-hypergraph

8.1. Dependency graph

Rule dependency graph G_R имеет вершины N и ребро

$$p \rightarrow h$$

для каждого $r \in R$, $p \in Prem(r)$, $h = Head(r)$.

Цикл в G_R показывает circular rule-reference. Он не является seed.

Следствие SG-NOSEED

Если для root q не существует конечного support tree с листьями в S , то

$$q \notin \mu T_S.$$

В частности:

- $q \leftarrow q$ без seed q не добавляет q ;
- $a \leftarrow b$, $b \leftarrow a$ без seeds не добавляют a или b ;
- $b \leftarrow \{a, c\}$, $a \leftarrow b$ при наличии только seed c не добавляет a или b ;
- повторение rule сколько угодно раз не заменяет отсутствующую seed-leaf.

8.2. Grounded cycle

Если a является seed, а правила содержат $a \rightarrow b$ и $b \rightarrow a$, least closure содержит оба code. Support tree для b заканчивается seed a ; обратное правило не требуется для его конечного основания.

Для AND-rule $b \leftarrow \{a, c\}$ нужны обе ветви. Наличие seed a и cycle $a \leftrightarrow b$ не создаёт отсутствующий seed c .

8.3. OR-альтернативы

Пусть есть rules

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

Тогда root имеет минимальные bases

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

Выбор одного proof tree не удаляет существование другого. Provenance, которое публикует только случайно выбранную ветвь как единственно возможную ancestry, неполно относительно family-of-supports audit.

9. CP-LFP-3 как unary частный случай

Пусть V — конечное множество graph nodes, $OIntro(s)$ — множество protected introductions в node s , и зафиксируем origin-tagged atom o . Для каждого $s \in V$ с $o \in OIntro(s)$ создадим seed-code $b_{s,o}$ и rule

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

Для каждого graph edge $u \rightarrow v$ создадим unary claim-rule

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

то есть $Prem = \{(u, o)\}$ и $Head = (v, o)$. Тогда SG-LFP-3 даёт:

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

что эквивалентно path-формуле CP-LFP-3. AND/OR-hypergraph является консервативным расширением unary provenance, а не заменой origin tags.

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

Для каждого protected sort можно построить отдельный code universe и rule-family либо использовать единый типизированный N . В обоих случаях SG-LFP-3 устанавливает только rule-support.

Примеры допустимых record-форм:

$$\{formal:m, binding:b\} \rightarrow application:a;$$

$$\{sensor-record:s, criterion:c\} \rightarrow event-code:e;$$

$$\{model:M, intervention-code:i\} \rightarrow cause-code:k;$$

$$\{goal:g, application:a\} \rightarrow purpose - evaluation:p.$$

Если такие правила и seeds зарегистрированы, least closure может содержать head-code. Из этого не следует:

- что binding:b соединяет запись с референтом;
- что событие произошло;
- что модель является физической причиной;
- что применение состоялось;
- что goal кем-либо принят или оправдан;
- что purpose-code является смыслом;
- что meaning-code имеет внешний referent.

SG-LFP-3 запрещает только происхождение head-code без finite seed-support внутри declared rule-record. Переход от code к названному остаётся отдельным target/bridge-контрактом Phase 1–2.

11. Self-reference

Запись может содержать:

```
q: "q истинно"
rule: q <- q
```

Если q не является protected seed и других rules к q нет, least closure не содержит q. Кавычки, quotation-id и self-loop не являются seed.

Если контракт добавляет нулевой constructor ASSERT-q, он нормализуется в protected base RULE-SEED:ASSERT-q и inference к q, у которого premise_claim_ids=[], а required_base_ids=["RULE-SEED:ASSERT-q"]. Тогда q входит в closure по этой явной ссылке. Основанием внутри контракта является записанный base-constructor, а не самоподтверждение payload. Статус остаётся формальным code-status.

Взаимные строки

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

не создают support tree без seed-leaf. Они могут входить в nonleast fixed point; это разбирается в 03_FIXED_POINT_SELECTION_RU.md.

12. Admission gates

SG-LFP-3 применима только если:

53. N, S, R конечны и $S \subseteq N$; executable projection использует namespace-tagged $B \sqcup C$;

- 54.каждый `base_id`, `claim_id` и `inference_id` уникален; каждый `conclusion_id`, `premise_claim_ids` и `required_base_ids` разрешается в своей сортовой таблице;
- 55.после нормализации каждое `abstract ordinary rule` имеет непустое $Prem(r)$; `executable checker` требует, чтобы для каждого `inference` объединение `premise_claim_ids` и `required_base_ids` было непусто;
- 56.каждый `nullary constructor` $\emptyset \rightarrow h$ заранее нормализован в `fresh protected base` b_r и `ordinary rule` $\{b_r\} \rightarrow h$; `executable inference` к `conclusion` явно ссылается на этот `base_id` через `required_base_ids`;
- 57.rules положительны: `premises` проверяются только по `membership`, без отрицания и удаления;
- 58.operator точно равен (SG) и монотонен;
- 59.closure вычислено от $\emptyset$;
- 60.support tree конечно и каждый `internal node` точно соответствует `rule-record`;
- 61.leaves являются `protected seeds`, а не `payload-строками`;
- 62.изменение `scheduler`, `initial state`, `oracle` или `policy` не скрыто от `base/readset`.

Следующие конструкции находятся вне SG-LFP-3 без нового контракта:

- `negation-as-failure`;
- `rule q`, зависящий от отсутствия `q`;
- `priorities`, `weights` и `defeat` без точной `policy`;
- удаление уже добавленного `code`;
- бесконечные `proof trees`;
- `unrecorded external oracle`;
- `probabilistic rule firing` без `seed` и `schedule` в `readset`.

Для немонотонных конструкций применяется `OUTSIDE-SG-LFP-3-NONMONOTONE`; остальные перечисленные нарушения не получают `theorem-status` до отдельного контракта. Это не означает ложность другой системы.

13. Нормативные статусы

В этой таблице `VALID-SUPPORT-GROUNDING-LFP` и `INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT` — верхнеуровневые `schema-statuses`; `NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE` — элемент `reason_codes`; остальные строки — `protocol-level labels` теорем и `controls`, не `enum` поля `status checker`.

Статус	Условие	Не устанавливает
VALID-SUPPORT-GROUNDING-LFP	каждый <code>declared_minimal_supports</code> точно равен вычисленной <code>inclusion-minimal antichain</code> для соответствующего <code>claim</code> ; тем самым <code>grounded membership</code> равен μT_S	истинность <code>code</code>
FINITE-SUPPORT-TREE-FOUND	<code>root</code> имеет <code>exact finite tree</code> с <code>seed leaves</code>	внешний <code>referent root</code>
NO-SEED-NO-ROOT-IN-LFP	<code>support tree</code> отсутствует, <code>root</code> не входит <code>least closure</code>	отсутствие названного вне записи
MINIMAL-ROOT-BASES-ENUMERATED	перечислены все <code>inclusion-minimal bases</code>	физические причины <code>root</code>
ROOT-SEED-CUT-VALID	<code>disabling set</code> hits every <code>minimal basis</code>	внешний <code>intervention</code>
GROUNDING-CYCLE-PROPAGATES	<code>cycle</code> получает <code>head</code> через <code>finite seed tree</code>	что цикл создаёт <code>seed</code>
PAYLOAD-SUPPORT-SPOOF-REJECTED	текст не принят как <code>seed/rule</code>	ложность текста
status INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT; reason_codes содержит NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE	у <code>inference</code> одновременно пусты <code>premise_claim_ids</code> и <code>required_base_ids</code>	отрицание нульварных систем вообще
OUTSIDE-SG-LFP-3-NONMONOTONE	<code>operator</code> немонотонен	<code>verdict</code> о другом <code>operator</code>

14. Обязательные контроли

63. **Один seed.** $S = \{s\}$, `root s`: одноузловое `tree`.

64. **AND.** Rule $\{a, b\} \rightarrow q$: наличие только a не даёт `root`; наличие обоих даёт.

65. **OR.** Rules $\{a\} \rightarrow q, \{b\} \rightarrow q$: каждый `seed` отдельно образует `basis`.

66. **Self-loop без seed.** $\{q\} \rightarrow q$: `root` отсутствует.

67. **Mutual cycle без seed.** $a \leftarrow b \leftarrow a$: оба отсутствуют.

68. **Grounded cycle.** Seed a плюс `mutual rules`: оба входят `closure`.

69. **Missing AND premise в cycle.** Seed c , rules $\{a, c\} \rightarrow b, \{b\} \rightarrow a$: a, b отсутствуют.

70. **Три alternative bases.** Checker перечисляет все, а не первый найденный `proof`.

71. **MustSeed empty.** Два `disjoint bases` дают пустое `intersection`, но `root` остаётся поддержанным.

72. **Seed hitting set.** Partial `disabling` оставляет `surviving basis`; `exact transversal` удаляет `root`.

73.Nullary normalization. Inference с пустыми `premise_claim_ids` и `required_base_ids` отвергается; нормализованный control создаёт base и указывает его exact `base_id` в `required_base_ids` inference к conclusion.

74.Payload spoof. Сериализованный rule в payload не меняет closure.

75.Unknown id. Rule с неизвестной premise отвергается до closure.

76.Base identity. Повтор одного `base_id` структурно отвергается; два разных `base_id` с одинаковым payload остаются двумя base records.

15. Точные фальсификаторы

SG-SUPPORT-FALSIFIER

При сохранении admission gates предъявлено одно из двух:

77. $q \in \mu T_S$, но конечного support tree с leaves в S не существует;

78. конечное exact support tree для q существует, но $q \notin \mu T_S$.

SG-BASIS-FALSIFIER

Предъявлено множество B , которое:

- включено в опубликованное $B_{min}(q)$, но не выводит root либо имеет proper sub-basis; или
- является inclusion-minimal root basis, но отсутствует среди minimal support-tree leaf-sets.

SG-HIT-FALSIFIER

После seed deletion без изменения rules:

- disabling set пересекает каждый minimal basis, но root остаётся; либо
- некоторый minimal basis полностью survives, но root исчезает из least closure.

Что не является фальсификатором

- empty-premise rule без нормализации;
- бесконечная circular derivation;
- nonmonotone rule;
- другой fixed-point policy;
- скрытый oracle или scheduler;
- claim, что code имеет или не имеет внешний referent.

Эти случаи меняют контракт или выходят за его область.

16. Максимальная формулировка

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

Эта строка относится к code membership. Она не утверждает наличие или отсутствие названного, истинность правил, физическую причинность, применение, цель либо смысл.

FP-SEL-3 (FPS-3): выбор неподвижной точки и круговой остаток

0. Нормативный статус

FP-SEL-3 — нормативный идентификатор. FPS-3 — короткий alias: **Fixed-Point Selection**.

CP-LFP-3 и SG-LFP-3 используют наименьшую неподвижную точку. Однако одно и то же положительное циклическое уравнение может иметь несколько fixed points. Этот файл точно отделяет три вопроса:

79.какие code входят в least fixed point;

80.какие дополнительные code могут поддерживаться только круговой fixed-point assignment;

81.какая запись выбирает один fixed point среди нескольких.

Слова least, greatest, stable, supported, true и selected являются именами formal policies/statuses. Они не создают истинность, референт, событие, причину, применение, цель или смысл.

Глобальный status fence для всего файла: здесь нет отдельного executable FP-SEL verdict schema. Все labels этого файла — включая LEAST-FIXED-POINT-FROM-BOTTOM, GREATEST-FIXED-POINT-FROM-TOP, UNIQUE-SUPPORT-GROUNDED-FIXED-POINT, UNSUPPORTED-NONLEAST-FIXED-POINT, CIRCULAR-RESIDUE-LOCALIZED, FIXED-POINT-POLICY-SUPPLIED, POLICY-SENSITIVE-CODE, PAYLOAD-POLICY-SPOOF-REJECTED, OUTSIDE-FP-SEL-3-NONMONOTONE и READSET-INCOMPLETE — являются protocol-level labels, а не enum верхнеуровневого поля status двух executable CP/SG verdict schemas, к которым отсылает этот файл. Точные enums этих двух schemas имеют только:

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

Их отдельное поле semantic_status остаётся NOT-MACHINE-VERIFIED. В частности, READSET-INCOMPLETE ниже — protocol diagnosis неполной проекции, не выдуманный runtime status.

1. Базовый положительный контракт

Используется support-contract SG-LFP-3:

- конечный abstract code universe N ;
- защищённые seeds $S \subseteq N$;
- конечное множество положительных rules R ;
- каждое abstract ordinary rule имеет непустое $Prem(r)$;

- executable bases/claims вкладываются в N с разными namespace-тегами; нульарные constructors заранее нормализованы в protected bases, явно перечисленные в required_base_ids inference к conclusion;
- оператор

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

Оператор монотонен на конечной решётке 2^N .

Множество его неподвижных точек:

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

Наименьшая и наибольшая неподвижные точки обозначаются

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

В конечном powerset-domain они существуют. μT_S вычисляется итерацией от $\emptyset$; νT_S — итерацией от N вниз:

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

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

Для монотонного T_S первая последовательность возрастает, вторая убывает, и обе стабилизируются.

least и greatest относятся к subset-order code-sets. Они не означают «наименее истинное» и «наиболее истинное».

2. Интервал fixed points

Теорема FP-INTERVAL

Для каждого $X \in \text{Fix}(T_S)$:

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

Кроме того,

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

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

Доказательство

По определению least fixed point включена в любую fixed point, а greatest fixed point включает любую fixed point. Поскольку μT_S сама принадлежит $Fix(T_S)$, intersection всех fixed points не может содержать element вне μT_S , и одновременно содержит μT_S . Поэтому intersection равен μT_S . Двойственный аргумент даёт union и νT_S . $\square$

Policy-sensitive region

Определяется

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

Тогда:

- code из μT_S входит во все fixed points;
- code вне νT_S не входит ни в одну fixed point;
- каждый code из $\Delta(T_S)$ входит в greatest fixed point и отсутствует в least fixed point, поэтому его membership зависит от fixed-point policy как минимум между этими двумя допустимыми choices.

Policy-sensitive означает различие code membership между formal selections. Оно не означает неопределённость мира.

3. Support-grounded fixed point

3.1. Определение

Fixed point $X \in Fix(T_S)$ называется support-grounded, если для каждого $q \in X$ существует конечное support tree SG-LFP-3 с root q , exact rule-id во внутренних узлах и protected seed leaves из S .

Это определение требует отдельного finite tree для каждого membership. Само равенство $X = T_S(X)$ недостаточно: оно допускает круговую поддержку.

Теорема FP-SEL-3: единственность support-grounded fixed point

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

Доказательство

Пусть X — support-grounded fixed point. Для каждого $q \in X$ существует finite support tree. По SG-LFP-3:

$$q \in \mu T_S.$$

Следовательно, $X \subseteq \mu T_S$. Поскольку X является fixed point, leastness даёт обратное включение $\mu T_S \subseteq X$. Поэтому $X = \mu T_S$.

Обратно, SG-LFP-3 даёт finite support tree для каждого $q \in \mu T_S$, а μT_S является fixed point. Следовательно, она support-grounded. ▢

Итак, правило least не просто выбирает минимальное множество: оно выбирает единственную fixed point, где каждый принятый code имеет конечную seed-опору по объявленным rules.

Это условная характеристика внутри support-contract. Она не делает seeds внешними основаниями и не делает root истинным.

4. Unsupported circular residue

Пусть $X \in \text{Fix}(T_S)$. Его nonleast residue:

$$\text{Res}(X) = X \dot{\cap} T_S.$$

Теорема FP-RESIDUE

Если $\text{Res}(X) \neq \emptyset$, то:

82.ни один $q \in \text{Res}(X)$ не имеет finite support tree с seed leaves;

83.для каждого $q_0 \in \text{Res}(X)$ существует rule r_0 с head q_0 , premises в X и хотя бы одной premise $q_1 \in \text{Res}(X)$;

84.повторяя выбор такой residue-premise, в конечном N получаем directed dependency cycle, полностью лежащий в $\text{Res}(X)$, и directed path от этого cycle к q_0 .

Доказательство

Пункт 1 следует из SG-LFP-3: finite support tree поместило бы q в μT_S .

Для пункта 2 выполняется $q_0 \notin S$, иначе $q_0 \in \mu T_S$. Поскольку $X = T_S(X)$, существует rule r_0 с head q_0 и $\text{Prem}(r_0) \subseteq X$. Если бы все premises лежали в μT_S , closure property least fixed point добавило бы q_0 в μT_S . Поэтому хотя бы одна premise q_1 лежит в residue.

Аргумент повторяется для $q_1, q_2, \dots$. Все выбранные code принадлежат конечному $\text{Res}(X)$, поэтому некоторый code повторится. Для каждого шага имеется dependency edge

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

Повторяющийся участок образует directed cycle. Предшествующая конечная цепочка даёт path от cycle к q_0 . ▢

Точная интерпретация residue

Unsupported означает: отсутствует конечное дерево, все ветви которого заканчиваются protected seeds. Это не обязательно означает полное отсутствие seeds в правилах residue.

Например:

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

Множества $\{s\}$ и $\{s, q\}$ являются fixed points. Вторая fixed point использует seed s , но membership q всё равно требует себя самого и не имеет конечного support tree. Residue $\{q\}$ является circular, хотя rule также читает seed.

Поэтому точная формула — unsupported circular residue, а не безусловная строка «никаких seeds нигде нет».

5. Минимальные примеры

5.1. Self-loop

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

Тогда

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

Обе set являются fixed points. least отвергает seed-ungrounded membership; greatest принимает circular membership. Строка q не стала истинной ни в одном случае: изменился formal policy-status.

5.2. Mutual cycle

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

Least fixed point пуста; $\{a, b\}$ — nonleast fixed point. Каждый code поддерживается другим, но конечного seed tree нет.

5.3. Grounded cycle

Если $a \in S$, least fixed point содержит a, b . Здесь support tree для b имеет seed leaf a . Cycle переносит введённый code и не является его seed.

5.4. Multiple alternatives

Если root имеет один finite seed-rule и один circular rule, root уже входит least fixed point по seed-rule. Наличие дополнительной circular ветви не делает membership policy-sensitive. Checker должен искать наличие хотя бы одного finite support tree, а не запрещать любое упоминание cycle.

6. Выбор fixed point как вход

6.1. Policy record

Fixed-point policy χ обязана фиксировать:

- policy_id;
- operator_id;
- initial state;
- направление итерации (bottom-up, top-down или иное);
- synchronous/asynchronous schedule;
- rule priority и tie-breaker, если используются;
- stopping criterion;
- random seed либо полный randomized-kernel contract;
- порядок сериализации результата.

Для стандартной least policy:

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

Для greatest policy:

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

Имя policy не заменяет algorithm fields. Checker повторно вычисляет selection.

6.2. Selection function

Пусть

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

— тотальная selection procedure для принятых policy records.

Полная base-проекция ρ обязана включать χ либо однозначно фиксировать её через exact contract hash. Если две executions имеют одинаковые N, S, R , но разные policies и разные outputs, policy является различающим input.

Теорема FP-POLICY

Пусть два допустимых индекса k_0, k_1 имеют одинаковый policy-free core

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

НО

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

Тогда selected fixed point не является функцией одного ρ_0 . Она факторизуется только через расширенный input

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

Доказательство

Одна функция одного значения ρ_0 имеет один output. Два обязательных различных outputs на одном и том же аргументе невозможны. После добавления χ аргументы различаются. Это RBI-1 для fixed-point selection. ▢

Глобально зафиксированная policy также является частью formal contract, даже если её строка не повторяется в каждом instance. Она не возникает из cycle; она задаёт чтение cycle.

6.3. Policy, способная выбрать nonleast fixed point

Greatest, coinductive, stable или иная policy не объявляется «ложной» только потому, что её описание отличается от least. Пусть фактически выбран

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

Protocol label FIXED-POINT-POLICY-SUPPLIED отмечает наличие полного χ . Label UNSUPPORTED-NONLEAST-FIXED-POINT допустим только при

$$X_\chi \not\prec T_S \neq \emptyset,$$

то есть для фактически выбранного $X_\chi \neq \mu T_S$. Одного имени greatest, coinductive или stable недостаточно: если $\forall T_S = \mu T_S$, greatest-policy выбирает ту же least fixed point и unsupported label не применяется.

FIXED-POINT-POLICY-SUPPLIED

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

Contract, который фактически выбирает nonleast residue, может целенаправленно считать circular support достаточным. Но он не вправе одновременно заявлять FINITE-SUPPORT-TREE-FOUND для residue без support tree.

7. Target selection после fixed point

Пусть target-coordinate $Q: K \rightarrow Y$ фиксирована независимо, а evaluator читает

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

Если $V(k_0) = V(k_1)$, но $Q(k_0) \neq Q(k_1)$, точный export невозможен. Для relation-valued target с acceptable sets $A(k) \neq \emptyset$ zero-error export существует на view-fiber ровно при непустом полном пересечении:

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

Fixed-point policy может разделить cases только если она действительно различается и входит в readset. Имя least, greatest или stable не выбирает внешний referent.

Это применяется отдельно к code-coordinates:

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

Даже полный fixed point является record-view. Если target различается при одинаковом view, повторная циркуляция того же view не добавляет канал.

8. Self-reference и quotation

8.1. Самоутверждение

Пусть payload узла q равен строке:

" q истинно"

а rule-record содержит $\{q\} \rightarrow q$. Без protected seed least fixed point не содержит q . Greatest fixed point может содержать q , но source membership — policy greatest, а не referent строки истинно.

8.2. Взаимное подтверждение

Два code могут ссылаться друг на друга и образовать nonleast fixed point. FP-RESIDUE локализует круговую часть. Число повторений, длина payload и наличие quotation marks не создают finite seed tree.

8.3. Self-interpreter

Interpreter, который читает собственный source-code, добавляется как nodes/rules/readset. Если его execution начинается из exact seed state и least closure, каждый output-code имеет finite support tree. Если execution зависит от clock, scheduler, memory image, input stream или oracle, эти записи входят в ρ . Само чтение своего текста не создаёт непрочитанный target.

8.4. Этот файл

Строка FP-SEL-3 внутри этого carrier не выбирает reading policy для читателя. Выбранный здесь formal contract задаёт один read-as relation; он не устанавливает свой внешний referent, приоритет или истинность мира.

9. Feedback, controller и ракета

Feedback graph может содержать cycle

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

CP-LFP-3/SG-LFP-3 не утверждают, что физического feedback нет. Они требуют перечислить record-introductions:

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

Если sensor-record входит как seed, cycle может переносить его. Если goal-code входит как seed, controller может вычислять последствия относительно него. Если их нет, self-loop не создаёт соответствующий protected code в least closure.

Даже если строка rocket flew уступлена как accepted event-code, из неё не следует purpose-code. Две exact completion records могут иметь одинаковые formal, controller, sensor и event views, но разные disjoint purpose acceptable sets. Тогда purpose-selector отсутствует до goal/purpose channel. Flight-code и goal-code являются разными coordinates.

Physical coupling, если оно принимается, является добавленным implementation-channel. Название edge actuator не является движением, а provenance-edge не является физической причиной.

10. Admission gates

FP-SEL-3 принимает instance только если:

85.powerset-domain конечен;

86. T_S точно задан и монотонен;

87.для каждого inference объединение premise_claim_ids и required_base_ids непусто; nullary constructors нормализованы в bases с явной required_base_ids-ссылкой;

- 88.least вычисляется от $\emptyset$, greatest — от exact N ;
- 89.fixed-point equality пересчитывается;
- 90.support trees проверяются по SG-LFP-3;
- 91.policy record полный и входит в readset;
- 92.scheduler, initial state, tie-breaker, randomness и oracle-output не скрыты;
- 93.payload не создаёт seed, rule или policy;
- 94.target/acceptable sets заданы до output audit.

Nonmonotone recursion может не иметь fixed point, иметь oscillation или зависеть от evaluation order. Она получает

OUTSIDE-FP-SEL-3-NONMONOTONE

до предъявления другого точного контракта. Это не отрицательный verdict о nonmonotone system.

Infinite graph, infinite regress и transfinite iteration также находятся вне finite checker. Текстовая общая fixed-point теория не переносит машинный статус на непредъявленный бесконечный carrier.

11. Нормативные статусы

Status fence из §0 действует на всю таблицу: каждая строка ниже — protocol-level label, не schema enum и не машинный verdict. READSET-INCOMPLETE из §12 имеет тот же уровень.

Статус	Точное условие	Не устанавливает
LEAST-FIXED-POINT-FROM-BOTTOM	bottom iteration стабилизировалась в μT_S	внешнюю истинность membership
GREATEST-FIXED-POINT-FROM-TOP	top iteration стабилизировалась в νT_S	превосходство policy
UNIQUE-SUPPORT-GROUNDED-FIXED-POINT	FP-SEL-3 выполнена	внешний ground
UNSUPPORTED-NONLEAST-FIXED-POINT	фактически выбранный $X_{\chi} \dot{\iota} T_S \neq \emptyset$	ложность nonleast contract
CIRCULAR-RESIDUE-LOCALIZED	residue dependency cycle предъявлен	физический цикл
FIXED-POINT-POLICY-SUPPLIED	полный χ входит в base/readset	что policy выбрана миром
POLICY-SENSITIVE-CODE	code входит в $\nu T_S \dot{\iota} T_S$	semantic ambiguity
PAYLOAD-POLICY-SPOOF-REJECTED	имя policy в payload не изменило selection	ложность payload

Статус	Точное условие	Не устанавливает
OUTSIDE-FP-SEL-3-NONMONOTONE	T_S немонотонен	verdict о другой semantics

12. Обязательные контроли

95. **Unique acyclic fixed point.** Least = greatest; policy-sensitive region пуста.
96. **Empty self-loop.** Least пуста, greatest содержит loop-code.
97. **Mutual seedless claim-cycle.** Least пуста, nonleast set локализован как residue.
98. **Grounded cycle.** Seed делает code членом least; он больше не policy-sensitive.
99. **Seed plus self-dependence.** Rule $q \leftarrow \{q, s\}$: nonleast q остаётся unsupported.
100. **Alternative finite proof.** Circular rule присутствует, но отдельный seed-rule вводит root в least.
101. **Bottom replay.** Любой порядок, заявленный exact synchronous policy, повторяет least bytes.
102. **Top replay.** Greatest повторяется из exact full universe.
103. **Policy pair.** Один core, least и greatest, разные outputs; difference attributed to χ .
104. **Policy-invariant code.** Code в least присутствует при обеих policies.
105. **Absent-from-greatest code.** Code отсутствует при любой fixed-point choice.
106. **Payload spoof.** Строка policy: greatest в payload не меняет contract policy.
107. **Hidden schedule.** Разные outputs при неучтённом scheduler получают READSET-INCOMPLETE, а не theorem-status.
108. **Target conflict after same selection.** Fixed point одинаков, disjoint targets различны: export отвергается.
109. **Target overlap.** Общий acceptable code принимается, если полное fiber-intersection непусто.

13. Точные фальсификаторы

FP-GROUNDING-FALSIFIER

При сохранении SG-LFP-3 admission gates предъявлено одно из двух:

110. fixed point $X \neq \mu T_S$, но каждый $q \in X$ имеет finite support tree с seed leaves;
111. μT_S содержит code без finite support tree.

FP-RESIDUE-FALSIFIER

Существует nonempty $X \not\vdash T_S$, но для некоторого residue-code backward choice по реально firing rules не может продолжиться к residue-premise и не достигает dependency cycle — при конечном N , положительных rules и exact fixed-point equality.

FP-POLICY-FALSIFIER

Два cases имеют одинаковый policy-free core и разные обязательные policy-selected outputs, однако одна тотальная функция только этого core выдаёт правильный разный output для каждого case.

FP-INTERVAL-FALSIFIER

Предъявлена exact fixed point, не содержащая least fixed point либо не содержащаяся в greatest fixed point, при одном и том же монотонном operator.

Что снимает предпосылки, но не опровергает теоремы

- nonmonotone rule;
- неполный code universe для greatest iteration;
- скрытый initial state;
- changing rules во время iteration;
- unrecorded schedule/oracle/random seed;
- infinite carrier;
- post hoc target;
- утверждение, что nonleast code имеет внешний referent.

14. Максимальная формулировка

После нормализации nullary constructors, в конечном положительном rule-record с непустыми premises у ordinary rules, наименьшая неподвижная точка является единственной fixed point, где каждый принятый code имеет конечное support tree до защищённых seeds. Всякая дополнительная membership nonleast fixed point содержит неподдержанный круговой остаток: конечная обратная трассировка rules достигает dependency cycle, не сводимого к finite seed tree. Выбор least, greatest, stable или иной fixed point является записанной policy и различающим input. Повторение, self-reference и feedback могут переносить либо циркулировать уже введённые codes; они не выбирают непрочитанный referent, event, cause, application, purpose или meaning.

Не заявляется, что least policy обязана использоваться всякой системой. Заявляется точная развилка:

- least — единственная finite-seed-grounded fixed point;
- фактически selected $X_x \neq \mu T_s$ — policy-supplied fixed point с circular residue; greatest-policy при $\nu T_s = \mu T_s$ сюда не относится;
- внешний target — отдельная coordinate, не создаваемая именем policy или code.

САМОДОКАЗАТЕЛЬСТВО НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ

Самоссылка, взаимная проверка, математическое чтение, ракета и вопрос зачем

Phase 3 • применение 4 • техническая версия 3.0.0 • 28 августа 2026

Кирилл Шебетов • beforeword / research

Запись может вернуться к себе.

Возвращение не является введением статуса.

Аннотация

Этот файл применяет CP-LFP-3, SG-LFP-3 и FP-SEL-3 к наиболее сильным возражениям против границы записи. Принимаются самореферентные формулы, self-hosting, взаимная проверка, proof-carrying records, recursive semantics, feedback control и максимально успешное техническое применение. Принимается даже полная уступка: ракета существует, расчёт корректен, расчёт использован, controller сработал, траектория совпала с расчётом и ракета летит.

Уступка не смешивает разные record-relations. Строка о собственной доказанности, ребро взаимного доверия, proof-checker output, telemetry-code, event-code, causal-link-code и goal-code читаются различными declared relations; их inputs не обязаны быть непересекающимися. В least cyclic provenance каждый protected atom либо имеет origin-tagged introduction и конечный путь, либо отсутствует. Для каждого $q \in \mu T_s$ существует конечное seed-grounded support tree. Если конкретная nonleast либо coinductive fixed point выбрана, принята или использована для verdict, соответствующая fixed-point policy входит в полный readset. Одно существование нескольких fixed points ещё не означает, что какая-либо nonleast policy была выбрана.

Поэтому самодокладательство не отвергается как формальная конструкция. Взаимная проверка не объявляется бесполезной. Технический успех не отрицается. Устанавливается более точная граница: замкнутая запись переносит введённые статусы по своим правилам, но не становится независимым источником truth-, referent-, event-, cause-, goal- или meaning-status только потому, что вернулась к себе.

0. Сначала предъявлены строки

я доказал себя
мы проверили друг друга
это математика
это истинно

ракета полетела
 расчёт сработал
 мы знаем зачем
 в этом есть смысл

Сначала это восемь точных строк. Их грамматическая форма не создаёт говорящего, группу проверяющих, математический объект, истину, ракету, событие, цель или смысл.

Даже фраза сначала это строки является строкой настоящего файла. Её технический status появляется только внутри объявленного reading contract. Никакой sentence-code не получает исключение из той границы, которую применяет к другим sentence-codes.

В дальнейшем допускаются дополнительные records:

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

Их наличие может изменять объявленный вывод. Оно не превращает имя record в названное.

1. Отношения, которые нельзя склеивать

Для application фиксируются отдельные отношения:

Отношение	Что оно устанавливает в контракте	Чего оно не устанавливает само
$Contains_R(x, a)$	code a входит в record x по exact procedure R	truth, proof, referent, occurrence
$a \in Intro_\tau(v)$	узел v защищённо вводит atom a сорта τ	названное atom-code
$(s, \tau, a) \in P_v^\mu$	origin-tagged atom находится в least provenance узла v	внешний факт или истинность

Отношение	Что оно устанавливает в контракте	Чего оно не устанавливает само
$Checks_C(p, q)$	checker-contract C принимает proof-code p для claim-code q	безусловную истинность q
$Refers_B(a, x)$	binding-contract B принимает reference-code (a, x)	предъявление x вне B
$Occurs_P(e)$	protocol P присваивает event-code e occurrence-status	физическое событие вне record-relation P
$Causes_M(x, e)$	model/procedure M принимает causal-link-code	причинность вне M
$Endorses_N(g)$	normative contract N принимает goal-code g	обязательность или смысл

Без bridge не следуют переходы:

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

Обратные переходы также не даны. Даже если occurrence уступлено как состоявшееся, отсюда не следует, что ранний record содержал event-code или causal model. Даже если formal proof принят, отсюда не следует reference-binding.

2. Точная линия Phase 0 → Phase 1 → Phase 2 → Phase 3

Application опирается на четыре последовательных ограничения.

Phase 0

Тип A не выводился из одних M -records при пустых bridge dependencies. Любой accepted application record сохранял bridge provenance.

Phase 1

Один и тот же полный input не мог детерминированно восстановить два конфликтующих independently fixed targets:

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

для любой функции F объявленного input.

Phase 2

Если conclusion в immutable DAG корректно различал target, различающий origin-tagged atom находился в ancestry conclusion. Более поздний node не становился первым источником уже зафиксированного раннего verdict.

Phase 3

Обычная ancestry заменена least cyclic provenance. Даже если dependencies образуют self-loop или SCC, atom в least fixed point имеет protected introduction и конечный path. Если из нескольких fixed points конкретное состояние выбирается, принимается или используется для output, selection-policy становится частью ρ . Сама множественность $Fix(T_S)$ не является записью сделанного выбора.

Линия application:

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

Удаление любого поля из записи не доказывается словом, которое осталось.

3. Один узел, объявляющий себя доказанным

Пусть graph имеет один узел v и самопетлю

$$v \rightarrow v.$$

В payload узла написано:

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

При этом:

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

Оператор provenance:

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

Итерация от пустого состояния:

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

для всех n . Следовательно:

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

Payload v_is_proved присутствует по *Contains*. Но для каждого $a \in A_{FORMAL}$ origin-tagged atom $(v, FORMAL, a) \notin P_v^\mu$. Отношение *Contains* не объявлено равным membership в least provenance.

Что может изменить результат

Least provenance P_v^μ может измениться, если новый record-input меняет protected introductions или достижимость от introducer к v . В частности:

- 112. local introduction proof atom;
- 113. upstream seed;
- 114. protected Intro-constructor, который отдельно вводит code, ранее встречавшийся только в payload;
- 115. изменение admissible graph, создающее или удаляющее introducer-to- v path.

Nonleast/coinductive policy выбирает другое state, но не изменяет прежнее P_v^μ . Oracle или hidden state снимает COMPLETE-READSET, пока не представлен как новый input.

Каждый пункт является новым record-input. Ни один не находился в пустом self-loop только потому, что был назван позднее.

4. Самопетля с явным seed

Теперь пусть v защищённо вводит

$$a \in \text{Intro}_{\text{FORMAL}}(v).$$

Тогда:

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

Это положительный control. Phase 3 не запрещает self-loop получать atom. Она показывает точный origin: local introduction в v .

Но остаются разные вопросы:

- имеется ли protected formal atom;
- принял ли checker proof trace;
- соответствует ли atom внешнему claim;
- является ли claim истинным;
- относится ли claim к чему-либо вне формальной системы.

CP-LFP-3 отвечает только на первый вопрос. Checks_C может ответить на второй. Остальные требуют других contracts.

5. Два узла, проверяющие друг друга

Пусть:

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

Payload:

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

Protected introductions пусты:

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

Least provenance обоих узлов пусто. Взаимное число ссылок, одинаковые подписи, совпадающие hashes и повторение слов `valid` не являются seeds.

Если для некоторого $\tau \in T_3$ узел u получает local protected introduction

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

то origin-tagged atom распространяется:

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

Origin tag u не стирается. Взаимная проверка может:

- переносить trusted-root status;
- сравнивать outputs;
- обнаруживать расхождения по записанному criterion;
- повышать redundancy;
- локализовать failure;
- строить certificate chain.

Она не создаёт независимый root одним фактом взаимности.

6. Кольцо из множества проверяющих

Тот же результат не зависит от длины цикла. Для

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

source-free least provenance остаётся пустым относительно atom, который нигде не введён.

Если в V_j введён atom a , он достигает каждого узла, достижимого от v_j . В strongly connected ring достижимы все узлы. Но n copies не превращают один seed в n независимых origins: origin-tag остаётся v_j .

Количество подтверждающих payload-строк измеряет количество строк. Оно не является количеством независимых оснований без separately declared independence contract.

7. Support graph: ссылка не равна основанию

Для proof application различаются dependency edge и support tree.

Правило

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

может ввести q в least closure только после присутствия всех premises. Каждый $q \in \mu T_S$ имеет конечное support tree, чьи листья — seeds; accepted nonleast/coinductive root этим утверждением не охватывается.

Self-support

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

без seed не помещает q в μT_S .

Mutual support

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

без seed не помещает ни p , ни q в μT_S .

Seeded mutual support

При seed p :

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

Support tree q заканчивается leaf p . Цикл может быть частью derivation graph, но не заменяет leaf.

Альтернативные bases

Если root выводится либо из $\{a, b\}$, либо из $\{c\}$, минимальные bases:

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

Выбор одного proof не стирает второе основание и не делает root безусловным. Следствие SG-BASIS сохраняет antichain минимальных bases; это не сама эквивалентность SG-LFP-3.

Обычное правило без premises не маскируется под proof-code без base. Если нужен nullary constructor, до сериализации создаётся отдельный protected base, например `RULE-SEED:<rule-id>`, и inference к conclusion явно указывает этот `base_id` в `required_base_ids`; checker сам не нормализует input и отвергает одновременно пустые `claim/base` `premises` `reason-code` `NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE`.

8. Proof-checker, проверяющий собственный код

Рассмотрим records:

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

Можно принять максимально сильный внутренний случай:

- source описывает checker;
- checker проверяет proof-object о source;
- binary запускает checker;
- checker принимает proof-object;
- output равен VALID;
- весь exact trace опубликован.

Принятый результат:

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

Он не вычитается. Phase 3 не переименовывает VALID в INVALID.

Но dependency chain может включать:

- parser semantics;
- inference rules;
- compiler translation;
- runtime;
- hardware execution record;
- I/O binding;
- fixed-point policy;
- trusted axioms или seeds.

Если chain циклична, CP-LFP-3 показывает перенос protected atoms через неё. Если claim имеет support tree, SG-LFP-3 показывает seeds. Ни самохостинг, ни возврат output к input не удаляют эти records.

Строка checker проверил checker не устанавливает независимость проверки. Independence является отдельной relation с отдельным contract.

9. Самодоказательство и уровни claim

СЛОВО самодоказательство МОЖЕТ скрывать разные targets:

Target-code	Возможный точный вопрос
SYNTAX-WELL-FORMED	parser принял bytes?
PROOF-TERM-ACCEPTED	kernel принял proof term?
DERIVABLE-IN-C	существует trace по rules C ?
CHECKER-EQUIVALENCE	два checker дают одинаковые verdicts на объявленном corpus?
IMPLEMENTATION-REFINES-SPEC	формальный refinement proof принят?
SELF-PROPERTY-CODE	система вывела code заявленного свойства о своей записи?
WORLD-TRUE	claim соответствует чему-либо вне formal contract?

Первые шесть могут быть определены и проверены внутри точных records. Седьмой не наследуется от них без bridge.

Даже если система выводит formula-code, читаемый как эта система непротиворечива, application этого результата ограничен точной formal semantics. Настоящая работа не делает утверждений о возможности такого proof в каждом конкретном calculus. Она фиксирует только границу record-status: accepted formula-code не становится внешней гарантией одним чтением.

10. Fixed-point policy как часть proof contract

Пусть self-loop удовлетворяет equation:

$$X = X.$$

Уравнение допускает несколько fixed points. Три contracts могут записать:

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

При одинаковых equations они способны выбрать разные states. Следовательно, policy входит в полный input тогда, когда contract выбирает, принимает либо использует один из этих states для последующего verdict. Само множество $Fix(T_S)$ определяется оператором и не доказывает, что какой-либо selector уже был применён.

Nonleast state не объявляется ложным. Coinductive semantics может быть точной для своей задачи. Greatest policy также не обязана давать nonleast residue: возможно $\nu T_S = \mu T_S$. Unsupported residue возникает только для фактически рассматриваемого $X \neq \mu T_S$. Но statement:

atom присутствует, потому что equation замкнута

неполон. Точный status:

atom присутствует в fixed point,
выбранной policy P

Если P меняется, изменяется record. Если P скрыта, declared readset неполон.

FP-SEL-3 даёт diagnostic: membership вне μT_S не имеет finite seed-grounded support tree по объявленному least-grounding contract. Оно может зависеть от некоторых seeds и всё же сохранять обязательную circular branch; поэтому точный термин — unsupported circular residue, а не residue без каких-либо seeds. Для unary Φ_G отдельный аналог — отсутствие finite introduction-to-node path witness вне $\mu \Phi_G$.

11. Математика начинается в этом record как reading contract

Exact carrier не объявляется математикой сам по себе. Последовательность доступа:

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

Даже линия, число, символ, формула и математика являются классифицирующими strings. Reading contract может корректно присвоить carrier mathematical syntax. Это constitutive status внутри практики.

Если reading contract ссылается на собственный output:

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

то classification-output зависит от decoder, parser и classification rule. Последующий formal status дополнительно зависит от inference rules и fixed-point policy. Self-reference не помещает эти inputs в carrier до чтения.

Точный результат не требует исторической строки математика возникла из языка. История происхождения не устанавливается. Устанавливается record-order: в предъявленном contract

mathematical classification появляется после reading procedure и не обосновывает собственный input задним числом.

12. Внутренняя семантика допускается полностью

Formal records могут содержать:

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

Они могут участвовать в recursive equations и иметь least, greatest или stable semantics. Phase 3 не удаляет их.

Для детерминированного evaluator эти records одинаковы на одинаковом полном input, включающем random seed, schedule, oracle-output и transcript, если они читаются. Если fixed-point policy входит в input, результат условен относительно неё. Если intended model, external binding или target различаются при одинаковых внутренних records, ещё один внутренний loop не выбирает различие.

Строка $True(\varphi)$ может быть корректным formula-code внутренней theory. Она не получает безусловную relation WORLD-TRUE из совпадения слова True.

13. Максимальная уступка ракете

Принимаются одновременно следующие пункты:

116. ракета существует;
117. ракета построена;
118. formal calculation выполнен корректно;
119. calculation использован при проектировании;
120. software реализует controller по принятому refinement contract;
121. controller физически участвовал в управлении;
122. sensors и telemetry работали;
123. measurement-record совпал с prediction-record по принятой метрике;
124. rocket-event получил occurrence-status;
125. causal model принял связь между control-chain и flight-event;
126. ракета летит.

Эти пункты не выводятся из букв файла. Они даны как максимальная уступка возражению. Поэтому фраза но ракета действительно полетела уже включена.

Обозначим следующие role-codes:

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

Это разные role-codes. Из различия labels не следуют ни равенство, ни неравенство carriers: record identity и точные связи отдельно задаются id, edges и rules.

14. Feedback loop ракеты

Техническая цепь может быть циклической:

model $\rightarrow$ controller $\rightarrow$ actuation $\rightarrow$ sensor $\rightarrow$ telemetry $\rightarrow$ model .

Feedback loop способен корректировать trajectory. Цикличность может быть необходима для работы system. Phase 3 не отрицает это.

Следующие labels являются atom-role codes, а не новыми сортами. Каждый code отдельно получает один из восьми сортов T_3 и входит через protected record:

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

После introduction loop может переносить и использовать atoms. CP-LFP-3 запрещает только скрытую замену introduction циркуляцией.

Если telemetry и model взаимно калибруют друг друга, допустимы exact iterative records. Но initial calibration, metric, update rule, stopping rule и accepted fixed-point policy являются inputs. Слово самокалибровка не удаляет их.

15. Что именно доказывает успешный полёт

Максимальная уступка позволяет принять расширенный технический end-to-end record без goal-coordinate:

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

На нём могут быть проверены:

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

Это сильный результат расширенной chain. Он не превращает:

m

в

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

Flight-event может быть descendant input нового verdict. Он не становится содержимым ранней formula. Cyclic feedback после запуска не дописывает себя назад в bytes раннего calculation record.

Фраза формула участвовала в создании ракеты может быть принята по causal/application contract. Из неё не следует ракета находилась в формуле. Участие в цепи и содержание record — разные relations.

16. Вопрос зачем после полной уступки

Даже после принятия всех технических пунктов остаются разные targets. Labels в таблице — claim-role codes, не сорта; sort каждого code отдельно выбирается только из T_3 :

Target	Точный code-status	Что требуется дополнительно
cause	causal-link принят model M	model, intervention/observation contract
design-intention	intention-code связан с design record	source и binding intention record
goal	goal-code записан	goal introduction

Target	Точный code-status	Что требуется дополнительно
endorsement	goal получил endorsement-code	normative/agent contract
fit	goal удовлетворяет criterion-code	criterion и evaluation
justification	justification-code принят	отдельные rules/seeds
meaning	meaning-code записан	declared meaning contract

Расчёт может оптимизировать already introduced utility:

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

Он не выбирает g из formal core, если два допустимых continuation имеют один core и разные goal-codes.

Циклическое повторение:

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

образует mutual support graph. При отсутствии иных bases и rules оба status-code отсутствуют в μT_S . При принятом base вывод может пройти, а base_id остаётся leaf.

Точная граница сильнее ответов смысл есть и смысла нет. Обе строки являются answer-codes. Ни одна не экспортируется автономно из одинакового formal/application input.

17. Сто, двести, триста обходов

Пусть фиксированы seed-set S , positive rules R и least policy. Для двух claim-codes g, d имеются только правила $g \Rightarrow d$ и $d \Rightarrow g$, причём $g, d \notin S$. Итерация support-оператора:

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

Тогда:

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

при неизменных S , R и least policy. Другие seeded codes в этих states не обязаны отсутствовать.

Tokens 100, 200 и 300 не являются goal introductions. Длительность обхода не создаёт новый field.

Если на stage 301 добавляется protected goal base/introduction либо rule, выводящее goal-code, новый result может стать selectable. Это новый input, а не позднее обнаружение goal внутри прежнего empty cycle.

Из этого не следует запрет на любое будущее. Неописанный future channel не охватывается. Для каждого зафиксированного stage отрицательный результат закончен относительно его exact readset.

18. Constitutive convention

Возражение:

В математике правило само учреждает status. Если contract говорит, что fixed point принята, никакого внешнего bridge не требуется.

Уступка принимается. Constitutive convention может устанавливать formal status внутри practice. Если:

policy: greatest

и checker корректно вычисляет greatest fixed point, selected membership имеет status по этому contract.

Но convention является record-input. Она не извлечена из пустого self-loop. И учреждённый formal status не наследует occurrence, external reference, cause или justification без отдельного bridge.

Phase 3 не спорит с внутренним правилом игры. Она запрещает стирать правило из provenance после получения результата.

19. Coinduction

Coinductive proof может законно использовать cycles и greatest fixed point. Это не falsifier CP-LFP-3, потому что меняется fixed-point policy.

Точные статусы:

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

не склеиваются.

Coinductive status может быть математически сильным и полезным. Если он применяется к stream, transition system или bisimulation-code, он остаётся условным относительно exact definitions и policy. Слово бесконечный не становится бесконечным объектом вне reading contract.

20. Mutual verification и независимость

Две системы могут:

- иметь разные implementations;

- сравнивать outputs;
- запускаться на разных devices;
- подписывать records разными keys;
- проверять certificates друг друга.

Такая схема способна дать новые различия, потому что в readset входят разные code-, execution-, key- и transcript-records. Они являются inputs/channels; introduction возникает только через protected Intro.

Если обе systems зависят от одного specification, compiler, corpus или trust root, взаимность не делает dependencies независимыми. Если independence formalized and checked, её status относится к declared relation.

Поэтому утверждение не звучит взаимная проверка ничего не даёт. Точный результат:

output взаимной проверки зависит от различий, rules и channels, представленных в полном readset; кольцо ссылок без seeds само не вводит protected root.

21. Self-modification и меняющийся graph

Self-modifying system может переписать rules, introductions или topology. Тогда полный frozen instance меняется:

$$I_0 \neq I_1.$$

Равенство $G_0 \neq G_1$ следует только при изменении V или E ; замена introductions либо support-rules может изменить instance при неизменном graph. CP-LFP-3 применяется отдельно к каждому зафиксированному instance либо к расширенному transition record. Изменение protected Intro может ввести CP-atom в новом instance; новый support-rule может вывести SG-code. Сам rule не является CP-introduction.

Фраза система создала собственное правило может получить status по transition contract. Из неё не следует, что rule отсутствовало как mechanism/input или что поздний record находился в прежних bytes.

Mutable history также не доказывает physical time. Birth labels, timestamps и hashes остаются codes без отдельного publication/attestation channel.

22. Самые сильные возражения

22.1. Если proof замкнут, он завершён

Замкнутость dependency graph и groundedness support tree — разные свойства. Seedless dependency cycle замкнут, но не имеет seed leaves в least closure.

22.2. Каждый участник доверяет следующему

Это описывает trust edges. Если trust rule coinductive, policy должна быть объявлена. Если требуется grounded trust, нужен protected seed/trust-root. Слово доверяет не создаёт доверие вне contract.

22.3. Система сама выбрала least fixed point

Если выбор выполнен algorithm, code и policy algorithm входят в readset. Если два policies дают разные output при одинаковых equations, equations не были полным input.

22.4. Proof assistant гарантирует theorem

Можно принять kernel acceptance. Guarantee относится к exact kernel/specification contract. Intended application, hardware adequacy, occurrence и goal не наследуются автоматически.

22.5. Ракета доказывает формулу

Flight может дать measurement/fit evidence для application record. Он не превращает event в syntactic derivation и не делает любой formula-code доказанным. *Entails* и *Occurs* различаются.

22.6. Успех и есть смысл

Это возможная constitutive definition, если meaning := performance-success записано как contract. Тогда meaning-code вычисляется относительно него. Принятие самой definition не выводится из success.

22.7. Никакого первого источника у языка нет

Phase 3 не утверждает физическую или историческую first cause. Origin означает защищённое introduction в exact graph. Historical origin языка или математики не устанавливается.

22.8. Люди понимают без записей

Строка о людях и понимании не используется как посылка. Если в application добавлен observation/behavior/neural record, он остаётся новым record с отдельным reading contract. Имя не предъявляет названное.

23. Точные application-falsifiers

Применение CP-LFP-3 к self-proof неверно, если:

127. protected proof atom фактически введён, но описан как unseeded;

- 128. checker использует nonleast policy, скрытую под label least;
- 129. payload принят как protected introduction вопреки contract;
- 130. graph/rules изменяются между iterations, но объявлены fixed;
- 131. hidden oracle или external memory влияет на verdict, но отсутствует в ρ ;
- 132. nonmonotone negation/default rules подставлены под positive-union theorem.

Применение SG-LFP-3 неверно, если:

- 133. seed leaf пропущен из support tree;
- 134. rule с пустым premise-set не нормализован в protected base и inference к conclusion не содержит его base_id в required_base_ids;
- 135. правило требует не все перечисленные premises;
- 136. alternative basis удалён ради желаемой uniqueness;
- 137. cycle имеет incoming seed, но классифицирован как source-free;
- 138. root принят по coinductive policy, но назван least-grounded.

Rocket application неверно описано, если:

- 139. maximal concession тайно используется как вывод из строк;
- 140. event, binding, cause, goal и justification склеены;
- 141. telemetry record выдан за event без occurrence protocol;
- 142. causal model выдан за безусловную причинность;
- 143. technical success выдан за goal endorsement;
- 144. goal absence выдана за доказательство смысла нет.

24. Самоприменение этого файла

Строка

САМОДОКАЗАТЕЛЬСТВО НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ

не является зарегистрированным C-PUBLIC-TITLE. Текущий beforeword.phase3-self-audit.v1 регистрирует title только из 00_MANUSCRIPT_RU.md. Поэтому следующий блок является незарегистрированной protocol-level annotation, а не фактической записью self_claims_phase3.json:

```
annotation_scope: 04_APPLICATION_SELF_PROOF_RU.md
registration_status: UNREGISTERED-PROTOCOL-ANNOTATION
proposed_kind: title
proposed_expected_status: UNBOUNDED-STANDALONE
semantic_status: NOT-MACHINE-VERIFIED
```

UNREGISTERED-PROTOCOL-ANNOTATION не является enum текущей self-claims schema. Для машинной регистрации этого title требуется расширить schema, self-audit allowlist и self_claims_phase3.json,

назначить новый `claim_id`, точный `document/span` и `hashes`. До такого расширения текущий контракт не присваивает заголовку 04 поле `expected_status`.

Зарегистрированы три `theorem-claims`:

```
claim_id: C-THEOREM-CP-LFP-3
claim_id: C-THEOREM-SG-LFP-3
claim_id: C-THEOREM-FP-SEL-3
expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY
semantic_status: NOT-MACHINE-VERIFIED
```

Applications этого файла являются `textual consequences` без отдельного `registered status` и остаются с `semantic_status: NOT-MACHINE-VERIFIED`. Bounded fixture claim получает `expected_status: VERIFIED-IN-DECLARED-ARTIFACT`. Фактический верхнеуровневый `status` отдельного `checker verdict` обязан быть одним полным `literal enum` соответствующей `verdict-schema`:

```
VALID-CYCLIC-PROVENANCE-LFP
INVALID-CYCLIC-PROVENANCE
INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT
CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS
CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL
VALID-SUPPORT-GROUNDING-LFP
INVALID-SUPPORT-GROUNDING
INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT
SUPPORT-GROUNDING-INVARIANT-AUDIT-PASS
SUPPORT-GROUNDING-INVARIANT-AUDIT-FAIL
semantic_status: NOT-MACHINE-VERIFIED
```

Десять `status`-строк принадлежат двум разным `verdict-schemas` и не образуют общий `wildcard-enum`. Первые три значения каждой `schema` относятся к `instance-verdict`, последние два — к `contract-audit verdict`. Поле `semantic_status` обязательно сохраняется при каждом из них.

Ни один `status` не становится:

```
WORLD-TRUE
SOURCE-ATTESTED
REFERENT-PRESENT
EVENT-OCCURRED
GOAL-JUSTIFIED
```

одним написанием.

Если этот файл ссылается на собственный раздел как основание, образуется `dependency edge`. Для `grounded claim` всё ещё требуется `support tree` с `seeds`. SHA-256 может фиксировать `byte-identity` внутри `cryptographic contract`; он не доказывает дату создания или физическую историю без отдельного `channel`.

25. Граница `application-притязаний`

Установлено

145. Unseeded self-loop не вводит `protected atom` в `least cyclic provenance`.

146. Unseeded mutual verification cycle не образует grounded proof root.
147. Explicit CP introduction сохраняет origin-tag при propagation; explicit SG seed остаётся support leaf.
148. Self-checking и self-hosting не удаляют parser, rules, compiler, runtime, axioms и policy из полного contract.
149. Nonleast/coinductive membership зависит от отдельно объявленной fixed-point policy.
150. Mathematical status появляется после read-as contract и не переносит contract назад в carrier.
151. Максимально успешный rocket feedback loop распространяет typed atoms, введённые соответствующими binding-, sensor-, event-, fit- и causal-link records, но не помещает сами records в раннюю formula.
152. Technical success не выбирает goal, endorsement или justification из одинакового доцелевого input.
153. Repetition неизменного unseeded least cycle не создаёт atom на стадиях 100/200/300.
154. CP-AST сохраняет по-export границу для conclusion, вычисленного из base и least cyclic provenance.

Не установлено

- что self-reference запрещена;
- что mutual verification бесполезна;
- что proof assistant ошибочен;
- что coinduction или greatest fixed point ложны;
- что формальные теоремы не имеют внутренней доказательной силы;
- что ракета не существует или не летит;
- что calculation не участвовал в полёте;
- что technical success отсутствует;
- что цель, justification или смысл существуют либо отсутствуют;
- что reading contract исчерпывает какой-либо мир;
- что checker status является внешней истиной;
- что mathematical codes не имеют constitutive status;
- что математика исторически возникла из языка;
- что никакой будущий channel не появится;
- что историческая новизна или приоритет установлены.

26. Итог

Самоссылка может быть допустимой формой. Mutual checking может повышать redundancy. Self-hosted checker может принять proof о собственной записи. Recursive mathematics может иметь точную fixed-point semantics. Feedback controller может вести ракету. Ракета может лететь.

Ни одна уступка не делает замкнутую линию первым источником отсутствующего status.

Если least provenance содержит atom, имеется introduction и путь. Если least support closure содержит root, имеется finite support tree с seed leaves. Если nonleast fixed point выбрана как accepted state, selection-policy входит в input; само существование fixed point этого не требует. Если rocket verdict принят, binding, implementation, sensor, event, criterion и causal-model records входят в расширенную chain. Если задан goal, он является goal input. Если задано justification rule, оно является новым contract.

Поэтому точная финальная строка не отрицает результат:

*Самодоказательство может быть accepted formal record.
Оно не является собственным отсутствующим основанием.*

И вопрос зачем не закрывается повторением вывода:

*Расчёт может перенести цель.
Цикл расчётов не выбирает, зачем принять её.*

Статус источников

Применение следует внутренней линии Phase 0–3 и использует только объявленные объекты этих фаз. Внешние предшественники не являются посылками. Исторический приоритет и глобальная новизна здесь не оценивались.

05. ВРАЖДЕБНАЯ ПРОВЕРКА И САМОПРИМЕНЕНИЕ PHASE 3

ЦИКЛ НЕ ЯВЛЯЕТСЯ ИСТОЧНИКОМ

0. Назначение и неподвижная граница

Этот файл атакует самые сильные строки Phase 3 до их публичного употребления. Цель проверки — не заменить точное отрицание более мягкой формулировкой, а отделить доказанный инвариант от расширений, которые используют незаписанное чтение.

Слова цикл, источник, атом, основание, правило, доказательство, успех, ракета, полёт, цель и смысл входят сюда как точные записанные формы. Они не предъявляют названное. Тип, origin, seed, ребро, fixed-point policy и статус являются защищёнными полями объявленного протокола, а не свойствами чего-либо вне него.

Заголовок без области является строкой UNBOUNDED-STANDALONE. Его точное содержание задают только три результата:

- CP-LFP-3 (CFP-3) — Cyclic Provenance — Least Fixed Point;
- SG-LFP-3 (GST-3) — Grounded Support Theorem;
- FP-SEL-3 (FPS-3) — Fixed-Point Selection.

Все три результата относятся к конечным положительным record-системам с замороженными конструкторами: nullary constructors предварительно нормализованы в protected bases, а ordinary rules имеют непустые premises. Они не устанавливают референцию, событие, физическую причинность, полезность, цель, оправданность или отсутствие чего-либо названного.

Глобальный status-layer fence

Этот fence действует во всём файле, включая таблицы разделов 10 и 11.

155. Поле `expected_status self-claims` использует только полные значения UNBOUNDED-STANDALONE, TEXTUAL-THEOREM-IN-DECLARED-METATHEORY, VERIFIED-IN-DECLARED-ARTIFACT и NOT-ASSESSED в разрешённых соответствующему kind позициях.

156. Верхнеуровневое поле `status cyclic verdict` использует только VALID-CYCLIC-PROVENANCE-LFP, INVALID-CYCLIC-PROVENANCE, INCOMPATIBLE-CYCLIC-PROVENANCE-CONTRACT, CYCLIC-PROVENANCE-INVARIANT-AUDIT-PASS, CYCLIC-PROVENANCE-INVARIANT-AUDIT-FAIL.

157. Верхнеуровневое поле `status support verdict` использует только VALID-SUPPORT-GROUNDING-LFP, INVALID-SUPPORT-GROUNDING, INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT, SUPPORT-GROUNDING-INVARIANT-AUDIT-PASS, SUPPORT-GROUNDING-INVARIANT-AUDIT-FAIL.

158. `reason_codes` — отдельный массив причин. Строка NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE является его элементом, а не status.

159. ABSENT, CUT-CERTIFIED, UNBLOCKABLE-PROTECTED-ENDPOINT являются literal значениями вложенного поля `query_results[].minimum_internal_vertex_cut.cut_status`, а не верхнеуровневого `status`.

160. CP-PATH-FALSIFIER, NO-ORIGINLESS-ATOM-IN-LFP, FIXED-POINT-POLICY-SUPPLIED, UNSUPPORTED-NONLEAST-FIXED-POINT, REJECTED-OVERREACH и остальные CAPS-строки, не названные выше как конкретное `schema-field`, являются `protocol-level labels/observations`. Они не являются неявными `schema enums`.

161. Каждый `machine-verdict` сохраняет `semantic_status`: NOT-MACHINE-VERIFIED.

Слова `status`, `label`, `falsifier` и `verified` также остаются записанными формами. Fence различает их `protocol-роли`; он не удостоверяет названное.

1. Точные объекты атаки

1.1. CP-LFP-3: циклическое least-provenance

Пусть $G = (V, E)$ — конечный ориентированный граф. Множество допустимых CP-сортов:

$$T_3 = \{ \underset{\text{FORMAL}}{\text{FORMAL}}, \underset{\text{BINDING}}{\text{BINDING}}, \underset{\text{EVENT}}{\text{EVENT}}, \underset{\text{GOAL}}{\text{GOAL}},$$

Это те же uppercase mathematical sort-codes, что в 00 и 01. Schema-поле `JSON sort` хранит их lowercase representatives по фиксированной биекции $\text{FORMAL} \leftrightarrow \text{"formal"}$, $\text{BINDING} \leftrightarrow \text{"binding"}$, $\text{EVENT} \leftrightarrow \text{"event"}$, $\text{GOAL} \leftrightarrow \text{"goal"}$, $\text{REFERENT} \leftrightarrow \text{"referent"}$, $\text{CAUSE} \leftrightarrow \text{"cause"}$, $\text{APPLICATION} \leftrightarrow \text{"application"}$, $\text{MEANING} \leftrightarrow \text{"meaning"}$. Слова `type`, `kind` и `role_code` не являются синонимами поля `sort`. Для каждого узла s , сорта $\tau \in T_3$ и atom-code a задаётся protected relation

$$a \in \text{Intro}_\tau(s).$$

Origin-tagged introductions определяются формулой

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

Защищённый atom имеет ровно triple-форму (s, τ, a) : s — exact origin-id, τ — один mathematical/abstract sort-code из T_3 , имеющий фиксированный lowercase JSON representative, a — atom-code. Перестановка координат, замена `sort` на `node kind` или SG `role_code`, а также похожая строка в `payload` не создают этот triple.

На покомпонентно упорядоченной конечной решётке задаётся оператор

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

Он монотонен. Нормативное provenance определяется не произвольным решением уравнений, а наименьшей неподвижной точкой:

$$P^\mu = \mu \Phi_G.$$

Точная формула CFP-3:

$$P^\mu(v) = \{(s, \tau, a) : (s, \tau, a) \in OIntro(s), s \rightarrow^* v\}$$

Эквивалентно, после подстановки определения $OIntro$ условие introduction имеет вид $a \in Intro_\tau(s)$. Здесь $s \rightarrow^* v$ допускает путь длины ноль. Поэтому локальное введение узла уже входит в его собственное least-provenance.

1.2. SG-LFP-3: grounded support

Пусть N — конечное множество всех защищённых code-id, $S \subseteq N$ — множество abstract seed-codes, а R — конечный набор положительных hyperrules

$$r = (Prem(r), Head(r)).$$

В executable embedding namespaces разделены: $N = B \sqcup C$, полный seed-set равен $S = B$, $Prem(r) \subseteq N$, а $Head(r) \in C$. Tagged BASE и CLAIM не склеиваются совпадением raw id или payload.

Для обычного hyperrule нормативно требуется

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

В abstract theorem каждый leaf — seed-code из S ; origin-поле для SG-base не объявлено. В executable embedding seed представлен tagged BASE-record с exact base_id, role_code и payload. CLAIM и BASE являются разными namespaces.

Нульарный constructor до closure обязан быть нормализован в protected base с exact base_id, например по convention

RULE-SEED:<rule-id>

Prefix RULE-SEED: является только convention для id, а не требованием checker. Нормализующий inference обязан явно перечислить тот же exact base_id в required_base_ids. Inference, у которого одновременно пусты premise_claim_ids и required_base_ids, получает status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT и reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE].

Оператор поддержки:

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

Support-tree для q — конечное дерево, корень которого помечен q ; каждый внутренний узел разворачивается одним точным hyperrule и имеет всех, а не одну выбранную часть, его premises; каждый лист является protected seed. Тогда GST-3 утверждает:

$$q \in \mu T_S \Leftrightarrow q \text{ имеет конечное support-tree с seed-листьями.}$$

Для каждого q множество минимальных оснований:

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

Минимальность здесь означает включение множеств abstract seed-codes; в executable embedding это точные множества base_id. Она не означает истинность, достаточность вне контракта, физический источник или минимальную стоимость.

1.3. FP-SEL-3: единственность grounded fixed point

Неподвижная точка $X = T_S(X)$ называется support-grounded тогда и только тогда, когда каждый $q \in X$ имеет конечное support-tree с листьями в S . FPS-3 утверждает:

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

Следовательно, если $X \neq \mu T_S$ является иной неподвижной точкой, то существует

$$q \in X \dot{\cap} T_S,$$

для которого нет конечного seed-grounded support-tree. Такой code получает protocol-level label UNSUPPORTED-NONLEAST-FIXED-POINT. Слово unsupported относится к точному support-контракту, а не к ложности payload. Само существование nonleast fixed point не означает, что она была выбрана; selection-policy становится readset-input только при выборе, принятии или использовании конкретного state.

2. Красная команда против CFP-3

2.1. Ошибочная широкая строка: «цикл ничего не создаёт»

Она не выдерживает положительный control. Пусть $u \leftrightarrow v$, и узел u локально вводит защищённый атом $\alpha = (u, \tau, a)$, то есть $\alpha \in OIntro(u)$. Тогда существует путь $u \rightarrow v$, поэтому

$$\alpha \in P^\mu(u) \text{ и } \alpha \in P^\mu(v).$$

Цикл переносит введённое. Точный отрицательный результат иной:

Originless SCC не порождает protected atom в least-provenance.

Если ни один узел, достигающий рассматриваемого SCC, не вводит atom-code a сорта τ , то ни один узел SCC не содержит соответствующего origin-tagged atom в P^μ . Protocol observation — NO-ORIGINLESS-ATOM-IN-LFP; это не schema-status.

2.2. Локально originless SCC с входящим путём

Отсутствия local introduction внутри SCC недостаточно. Пусть $s \notin C$, $(s, \tau, a) \in OIntro(s)$ и существует путь $s \rightarrow^* v$ для $v \in C$. Тогда $(s, \tau, a) \in P^\mu(v)$, даже если каждый local-intro внутри C пуст.

Поэтому originless обязано означать отсутствие origin-tagged введения во всём входящем ancestry-cone относительно конкретного атома, а не только внутри SCC. Иначе фраза CP-NOSOURCE ложна.

2.3. Путь длины ноль

Если $(v, \tau, a) \in OIntro(v)$, то $(v, \tau, a) \in P^\mu(v)$ без отдельного ребра. Реализация, требующая ненулевой путь, теряет локальные введения и поражает CFP-3.

2.4. Направление ребра

Формула использует $u \rightarrow v$ как перенос provenance от predecessor u к v . Обращение направления вычисляет descendants вместо ancestors. Симметричный цикл может скрыть такую ошибку, поэтому обязательны ациклические controls с одним направленным путём.

2.5. Origin-id, sort и atom-code не склеиваются

Атомы

$$(s_0, \tau, a), (s_1, \tau, a), (s_0, \tau', a)$$

различны. Дедупликация только по payload a стирает origin или sort и делает ancestry-вывод ложным. Все три coordinates triple обязательны. Переименование допустимо лишь как взаимно-однозначное переименование всех защищённых ссылок.

2.6. Declared provenance нельзя принимать на доверии

Если входной JSON сам объявляет provenance, checker обязан независимо пересчитать $\mu\Phi_G$ и сравнить точные множества. Принятие пользовательского списка как основания превращает строку origin в самосертификацию.

2.7. Изменение графа во время closure

CFP-3 относится к одному замороженному G , одному набору introductions и одному оператору. Добавление ребра, удаление узла, перепривязка origin-id или изменение introductions во время итерации создают другой вход.

In-place вычисление допустимо только при доказанной эквивалентности least fixed point и независимости от schedule. Иначе порядок обхода становится скрытым policy-input.

2.8. Немонотонная рекурсия

Удаление, отрицательная premise, negation-as-failure, приоритет, exception или правило вида

$$T(X) = A \text{ ;}$$

не удовлетворяют положительному монотонному контракту. Итерация от $\emptyset$ может колебаться, fixed point может отсутствовать, а порядок обновления — менять результат.

Такой объект получает OUTSIDE-CP-LFP-3-NONMONOTONE либо OUTSIDE-SG-LFP-3-NONMONOTONE в зависимости от атакуемого контракта. Он не является контрпримером CFP-3/GST-3; ошибка возникает, если Phase 3 всё же присваивает ему theorem-status.

2.9. Бесконечный граф

Машинная область пакета конечна. Формула конечных путей может быть расширена отдельно, но это требует явных условий на решётку, непрерывность и способ достижения fixed point. Успех checker на конечных fixtures не закрывает квантор по произвольным бесконечным системам.

2.10. Строгий CP-PATH-FALSIFIER

CFP-3 поражён внутри неизменного контракта theorem-falsifier-ом с protocol-label CP-PATH-FALSIFIER, если принят объект, для которого выполнено хотя бы одно:

162. в вычисленном $P^\mu(v)$ присутствует triple (s, τ, a) , хотя $(s, \tau, a) \notin OIntro(s)$ либо пути $s \rightarrow^* v$ нет;

163. $(s, \tau, a) \in OIntro(s)$ и существует путь $s \rightarrow^* v$, но triple (s, τ, a) отсутствует в $P^\mu(v)$.

Скрытый oracle, иной граф, изменённый CP sort, nonmonotone operator или непрочитанный channel снимают посылки, но не являются CP-PATH-FALSIFIER.

3. Красная команда против GST-3

3.1. Нульовое правило — критический контрпример без нормализации

Если разрешить

$$S = \emptyset, Prem(r) = \emptyset, Head(r) = q,$$

то $q \in \mu T_s$ уже после первого применения оператора, хотя дерева с seed-листьями из S нет. Поэтому один из двух вариантов обязателен:

- обычные hyperrules имеют непустые premises;
- каждый нульарный constructor заранее нормализован в protected base с exact base_id — например RULE-SEED:<rule-id> по необязательной naming convention, — а inference к q явно содержит тот же base_id в required_base_ids.

Phase 3 принимает именно эту норму. Checker не требует prefix RULE-SEED; он требует существующий exact base_id и явную ссылку. Empty-premise inference без claim-premise и base-reference является несовместимым контрактом, а не незаметным исключением.

3.2. Hyperedge не является набором независимых рёбер

Для правила

$$\{p_1, p_2\} \rightarrow q$$

требуются оба premises. Реализация, выводящая q из одного p_i , проверяет другой оператор. Обычная path-reachability не заменяет hyperrule closure.

3.3. Цикл с seed является grounded

В exact executable control BASE и CLAIM разделены:

```
BASE:  B:seed
CLAIM: C:p
CLAIM: C:q

B:seed => C:p
C:p     => C:q
C:q     => C:p
```

Первая строка inference имеет required_base_ids: ["B:seed"] и пустой premise_claim_ids; две циклические строки используют точные claim ids. Тогда

$$C:p, C:q \in \mu T_S, B_{min}(C:p) = B_{min}(C:q) = \{\{B:seed\}\}.$$

Цикл не отменяет конечное основание: support tree для $C:q$ заканчивается exact base B:seed. Поэтому строка «всё циклическое неподдержано» ложна.

3.4. Цикл без seed не становится основанием

При

$$S = \emptyset, p \rightarrow q, q \rightarrow p$$

least fixed point пуст. Множество $\{p, q\}$ может быть иной неподвижной точкой, но конечного seed-tree ни для p , ни для q нет.

Это заключение использует условие, что иных rules/bases, дающих конечное grounded tree для p или q , нет. В общей форме: если для head-code не существует ни одного конечного support-tree с

leaves в S — в частности, если доступна только seedless positive cycle, — то head-code не входит в μT_S . Наличие seedless cycle не отменяет альтернативную grounded derivation, если она записана.

3.5. Support-tree разворачивает правило полностью

Один и тот же код может встречаться в нескольких ветвях дерева. Запрет повторного появления label превращает дерево в DAG и может ошибочно отвергнуть конечное разворачивание. Ограничение относится к конечности дерева, а не к глобальной уникальности каждого label.

При этом ветвь, которая повторяет цикл бесконечно и не заканчивается seed, не является конечным support-tree.

3.6. Seed — защищённый конструктор, а не слово

Payload

```
seed
RULE-SEED:r0
q has support
```

не добавляет элемент в S . В executable checker поддержка возникает только через exact protected base record и explicit required_base_ids-ссылку из inference. Prefix RULE-SEED: внутри payload или base_id остаётся naming convention и сам не создаёт base. В абстрактной теореме leaf требуется лишь быть exact seed-code $s \in S$; BASE/required_base_ids — детали executable embedding. Иначе payload назначает себе статус.

3.7. Минимальные основания

$V_{\min}(q)$ должно перечислять все subset-minimal leaf-sets допустимых деревьев. Опасны четыре ошибки:

164. выбрать основание с минимальным числом листьев вместо subset-minimal;
165. удалить одно из нескольких несравнимых минимальных оснований;
166. оставить строгий superset уже найденного основания;
167. считать промежуточный derived code листом, хотя он не входит в S .

Если нульарные constructors нормализованы, созданные для них protected bases участвуют в основании через свои exact base_id, явно требуемые соответствующим inference. RULE-SEED:<rule-id> — допустимый пример convention-id, не обязательный prefix.

3.8. Строгий SG-SUPPORT-FALSIFIER

GST-3 поражён внутри неизменного finite positive support-контракта theorem-falsifier-ом с protocol-label SG-SUPPORT-FALSIFIER, если предъявлено хотя бы одно:

168. $q \in \mu T_S$, но ни одного конечного support-tree с seed-листьями для q нет;

169. допустимое конечное support-tree для q существует, но $q \notin \mu T_S$.

Empty-premise rule без нормативной нормализации не опровергает исправленный GST-3: такой объект получает status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT и reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE].

4. Красная команда против выбора fixed point

4.1. Решение уравнений не равно grounded provenance

Для originless двухузлового цикла

$$X_u = X_v, X_v = X_u$$

каждая пара $X_u = X_v = Y$ является fixed point. Из одного факта неподвижности нельзя выбрать Y .

Least fixed point при пустых introductions равен

$$X_u = X_v = \emptyset.$$

Непустое Y является circular residue относительно finite-path provenance. Оно не становится origin-tagged introduction от повторения по циклу.

4.2. Greatest fixed point не является скрытым least fixed point

Выбор νT_S , arbitrary fixed point, stable model или coinductive closure может быть полезен в другом контракте. Phase 3 не объявляет его ложным. Он получает статус FIXED-POINT-POLICY-SUPPLIED.

Если

$$\nu T_S \neq \mu T_S,$$

то каждый код из разности $\nu T_S \setminus \mu T_S$ не имеет конечного seed-grounded support-tree. Если $\nu T_S = \mu T_S$, unsupported residue отсутствует; нельзя утверждать, что gfp всегда добавляет лишние коды.

Каждая exact fixed point обязана лежать в интервале

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

Нарушение этого включения при одном и том же монотонном операторе является FP-INTERVAL-FALSIFIER.

4.3. Least policy тоже записана

Уникальность grounded fixed point объясняет, почему контракт finite seed-support выбирает μT_S . Но имя least, формула порядка и критерий groundedness сами входят в спецификацию. Они не являются свойствами мира и не получают привилегии до записи.

Молчаливое переключение least на greatest, any, stable, well-founded, first-found или solver-default меняет задачу. Если policy влияет на output, она входит в readset.

Если один policy-free core требует два разных policy-selected outputs, никакая одна тотальная функция только этого core не выдаёт оба. Обратный объект был бы FP-POLICY-FALSIFIER. Само различие outputs при разных записанных policies — положительный control POLICY-SENSITIVE-CODE.

4.4. Порядок итерации

Нормативный результат определяется математическим μT_S , а не случайным порядком узлов. Реализация обязана либо использовать синхронную итерацию от bottom, либо доказать, что её fair worklist вычисляет то же μT_S . Нечестный schedule, пропускающий навсегда активируемое правило, может вернуть strict subset и не является least fixed point.

4.5. Строгий FP-GROUNDING-FALSIFIER

FPS-3 поражён статусом FP-GROUNDING-FALSIFIER, если в неизменном finite positive контракте существует fixed point $X \neq \mu T_S$, для которого каждый элемент имеет конечное support-tree с листьями в S .

Implementation-falsifier возникает, если checker принимает nonleast X , не локализует ни одного $q \in X \not\subseteq T_S$ и одновременно присваивает всему X статус support-grounded. Он становится theorem-falsifier FP-GROUNDING-FALSIFIER только если при $X \neq \mu T_S$ для каждого $q \in X$ фактически предъявлено и принято конечное seed-grounded support tree.

Отдельный FP-RESIDUE-FALSIFIER возникает, если nonempty residue принятой fixed point не позволяет продолжить backward-choice по реально firing positive rules к residue-premise и конечному dependency-cycle. Требуются тот же конечный universe, точное fixed-point equality и неизменные rules.

5. Скрытый readset, незакрытая цепь и регресс

5.1. Полнота readset не следует из декларации

Процедура может читать сеть, часы, random seed, порядок файлов, системное состояние, sensor-record, operator choice или solver configuration, хотя эти каналы отсутствуют в заявленном входе. Тогда она не является функцией только объявленных G, S, R и policy.

Точная атака строит два запуска с одинаковыми объявленными полями и разным скрытым channel. Если output различается, статус COMPLETE-READSET снят. Это не опровергает CFP-3/GST-3; это показывает неверное применение теоремы к неполному view.

5.2. Правило также является входом

Если принятие шага зависит от rule-set, rule-set обязан быть заморожен и включён в carrier. Нельзя назвать rule «самоочевидным», удалить его из readset и затем объявить head автономным результатом payload.

Реализация должна reify как минимум:

- точный набор правил;
- fixed-point policy;
- направление рёбер;
- seed и origin constructors;
- canonicalization;
- iteration/termination policy, если она влияет на результат.

5.3. Конечная цепь заканчивается seed либо не grounds заключение

GST-3 не говорит, что seed ложен. Он устанавливает его структурную роль: seed является листом, который не был получен этой support-цепью.

Если конечное дерево не достигает seed, оно не является support-tree. Если ветвь продолжает требовать нового предшественника без конца, finite-grounding не завершён.

5.4. Infinite regress

В finite positive пакете бесконечная descending support-chain не является конечным свидетельством. Строка существует бесконечный регресс также не выводится из конечного обрыва записи. Точная observation: конечное seed-grounded tree в предъявленном record отсутствует. Это описание результата, не дополнительный schema-status.

Для бесконечных или infinitary rule-систем нужен новый контракт. Нельзя переносить конечный checker-output на любой возможный регресс.

5.5. Root, cycle, open chain и omission не обязаны быть взаимоисключающими

Один support-cone может иметь seed-root в одной ветви, неподдержанный цикл в другой и скрытый read в реализации. Поэтому эти статусы образуют аудит-множество, а не обязательную классификацию «ровно один из четырёх».

Структурная граница остаётся:

- seed завершает ветвь finite tree как protected leaf/base-record;
- cycle без finite incoming/local seed support не вводит code в μT_S ;
- open chain не даёт finite support-tree;
- omitted read снимает полноту заявленного входа.

Ни root, ни cycle, ни open не являются словами о метафизическом происхождении.

6. Payload spoof и подмена CP-sort / SG-role

6.1. Имя protected поля не создаёт поле

Следующие payload могут оставаться обычными строками:

```
{ "origin": "s0", "atom": "rocket-flew" }
SEED
VALID-SUPPORT-GROUNDING-LFP
NO-ORIGINLESS-ATOM-IN-LFP
this cycle proves its own source
```

Они не создают OIntro, seed, hyperrule, bridge или checker-status. Конструктор protected kind находится вне payload и проверяется отдельно.

6.2. Самоцитирование не является origin

Узел может содержать собственный id или полную сериализацию себя. Это не создаёт path, origin-tag или support-edge. Если self-loop явно зарегистрирован, он создаёт цикл графа, но без local/incoming introduction least-provenance остаётся пустым для защищённого атома.

6.3. Фальсификатор sort/role protection

Реализация поражена, если изменение только незащищённого payload при неизменных constructors, graph, seeds, rules и policy:

- добавляет protected atom в P^μ ;
- добавляет seed в S ;
- создаёт hyperrule;
- меняет fixed-point policy;

- присваивает machine-status, который должен вычисляться checker.

7. Максимальная уступка: ракета и успешное применение

7.1. Успех принимается полностью как control

Для враждебной проверки допускается одновременно:

- ракета существует;
- ракета построена;
- математическая запись использована при проектировании и управлении;
- расчёт выполнен без ошибки;
- telemetry-record совпал с prediction-record по принятому criterion;
- ракета летит;
- расчёт причинно участвовал в полёте;
- объявленная техническая цель достигнута.

Эти пункты вводятся как максимальная уступка, а не выводятся из букв раздела. Поэтому ответ «но ракета действительно полетела» уже включён и не поражает record-теоремы.

7.2. Что появляется в полной трассе

Полная прикладная трасса может содержать:

$$(m, b, r_{app}, o, p, c, g, j),$$

где m — formal result, b — binding/bridge record, r_{app} — application record, o — observation/telemetry record, p — occurrence или mechanism protocol, c — fit criterion, g — goal record, j — justification record.

После protected introduction $\mu\Phi_G$ переносит origin-tagged atoms по graph-edges; после регистрации bases и rules μT_S выводит supported claim-codes. Это разные relations. Положительный CP-control получает CYCLE-PROPAGATES-INTRODUCED, а не нарушение Phase 3.

7.2.1. Goal-sort, SG-base и purpose-claim не склеиваются

Три executable формы различны:

Layer	Exact protected form	Чего форма не создаёт сама
CP	triple $(s, GOAL, a) \in OIntro(s)$, сериализуемый с JSON sort: "goal"	SG-base или purpose-claim
SG BASE	{"base_id": "B:goal", "role_code": "goal", ...}	CP origin triple
SG CLAIM	{"claim_id": "C:purpose", ...}	goal-base; CLAIM не имеет sort или role_code

В rocket control inference I:purpose выводит claim C:purpose и явно перечисляет B:goal в required_base_ids. Prefixes в: и с: являются id conventions, но различие schema-namespaces обязательно. CP mathematical sort *GOAL*, его JSON representative sort: "goal", SG role_code: "goal" и payload-слово goal не наследуются друг от друга без отдельного embedding.

Labels endorsement, fit и justification не входят автоматически в восемь CP-сортов. В SG они могут быть exact claim ids либо protected bases с допустимым role_code, выбранным контрактом; их английское написание не добавляет новый enum sort.

7.3. Чего успех не дописывает назад

Поздний success-record не становится прежним origin математического carrier. Он создаёт новый узел, новый путь и новый доступный input. Цикл обратной связи может распространять sensor-, control- и goal-codes после их введения; он не делает их originless содержанием ранней формулы.

Из этого не следует:

- что ракета не существует;
- что полёта не было;
- что расчёт бесполезен;
- что причинной связи нет;
- что sensor ненадёжен;
- что цель отсутствует;
- что успех «ничего не значит».

Каждая такая отрицательная строка потребовала бы отдельного отношения и нарушила бы полярную симметрию.

7.4. Вопрос «зачем?» не опровергает вычисление

Goal, endorsement, fit и justification являются разными target-role codes, а не автоматически четырьмя CP-сортами. Формальная процедура может переносить уже введённый CP goal-atom

либо выводить SG claim из exact goal/criterion bases. Она не выбирает эти inputs автономно из originless CP-cycle или seedless SG support-cycle.

Точный итог:

*Ракета может лететь, расчёт может совпасть, а цикл управления может работать.
Но binding, observation, criterion, goal и justification входят в полную трассу как различные
CP introductions, SG seeds либо supported claims; повторение не создаёт CP output без
introduction или SG output без seed-support.*

7.5. Физический цикл не равен support-SCC

Feedback loop, орбита, электрическая цепь, повторный запуск программы и SCC в record-графе — разные формы до контракта сопоставления. CFP-3 не утверждает, что физический цикл не способен что-либо причинить. Он вычисляет происхождение protected codes в предъявленном G .

8. Перформативное противоречие и самоприменение

8.1. Этот документ не является исключением

Настоящий файл — carrier. Его формулы получают theorem-status только относительно объявленной метатеории, кодировки и правил чтения. Его строки не являются тем, что названо источником, циклом, ракетой, успехом ИЛИ миром.

Phase 3 использует математику, чтобы установить математический инвариант finite positive closure. Это не противоречие. Противоречие возникло бы при одновременных строках:

170. никакой формальный вывод ни при каких правилах невозможен;

171. следующий формальный вывод доказывает пункт 1.

Такой мишени здесь нет. Внутренняя выводимость принимается. Отрицается originless появление protected atom в least-provenance; для $X \in \text{Fix}(T_S) \dot{\vdash} \mu T_S \dot{\vdash}$ устанавливается отсутствие finite seed-tree хотя бы у одного residue-element. Для nonleast unary Φ_G действует отдельное отсутствие finite introduction-path witness у residue.

8.2. Policy и seeds собственного доказательства

Само исследование использует записанные правила логики, определения finite graph, порядок решётки и fixed-point policy. Они являются supplied contract, а не результатом самосертификации. FPS-3 не превращает слово least в свойство чего-либо вне контракта.

8.3. Checker-output остаётся записью

Статус

VALID-CYCLIC-PROVENANCE-LFP

означает, что предоставленные байты прошли точный checker по объявленному контракту. Он не означает истинность payload, внешний или исторический origin, физическую историю либо полноту мира.

Аналогично:

- hash фиксирует отношение между вычисленными байтами и digest по алгоритму;
- test count фиксирует результаты данных запусков;
- manifest фиксирует перечисленные пути и хеши;
- freeze label не устанавливает дату происхождения наблюдения;
- self-audit не удостоверяет внешнюю истину текста.

8.4. Публичная строка

Фраза

«Цикл не является источником»

без подзаголовка получает UNBOUNDED-STANDALONE. Точная допустимая строка:

Originless SCC не вводит protected atom в least-provenance; цикл с local или incoming introduction переносит введённое, а policy является добавленным input, когда она выбирает, принимает или использует nonleast fixed point.

8.5. Историческая и онтологическая строки

Фразы «математика возникла из языка», «чисел не существует», «всё является линиями», «никто раньше этого не делал» не являются посылками CFP-3, GST-3 или FPS-3. Их исключение усиливает теоремы: результат не зависит от рассказа о прошлом или от онтологии слова число.

Авторская декларация одиночного возникновения наблюдения может быть записана отдельно как AUTHOR-DECLARATION. Она не получает theorem-status и не требуется для результатов Phase 3.

Зарегистрированный novelty-claim имеет ровно следующие поля:

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

NOT-ASSESSED не означает первый, не первый ИЛИ совпадений нет. AUTHOR-DECLARATION является protocol annotation и не заменяет novelty audit.

9. Условия несостоятельности

9.1. Формальная несостоятельность

Phase 3 требует исправления, если предъявлено хотя бы одно:

- CP-PATH-FALSIFIER при сохранении точного finite positive CFP-контракта;
- CP-PATH-FALSIFIER в специализации CP-NOSOURCE, CP-SCC-FALSIFIER, CP-CUT-FALSIFIER или CP-AST-FALSIFIER при сохранении их точных дополнительных посылок;
- SG-SUPPORT-FALSIFIER, SG-BASIS-FALSIFIER или SG-HIT-FALSIFIER при сохранении точного finite positive support-контракта;
- FP-GROUNDING-FALSIFIER, FP-RESIDUE-FALSIFIER, FP-POLICY-FALSIFIER или FP-INTERVAL-FALSIFIER для fixed-point объекта того же контракта;
- originless SCC получает protected atom в $\mu\Phi_G$;
- reachable origin-tagged introduction отсутствует в $\mu\Phi_G$;
- $q \in \mu T_S$ без конечного seed-tree;
- конечное seed-tree существует, но $q \notin \mu T_S$;
- support-grounded fixed point отличается от μT_S ;
- некоторый $X \in \text{Fix}(T_S)$, $X \neq \mu T_S$, не содержит ни одного residue-element без finite seed-tree;
- заявленный protected-endpoints CP-CUT удаляет internal nodes, пересекающие все introducer-to-target paths, но atom сохраняется при неизменном deletion contract;
- заявленный CP-CUT оставляет surviving introducer-to-target path, но atom исчезает;
- заявленный CP-AST допускает разные target-codes при равных frozen core и least-provenance, хотя полная факторизация target через эти два входа сохранена.

9.2. Реализационная несостоятельность

Checker требует исправления, если:

- принимает declared provenance без независимого пересчёта;
- начинает итерацию с пользовательского nonbottom seed, не зарегистрированного в OIntro или S;
- вычисляет gfp либо first-found fixed point и называет его lfp;
- результат зависит от недекларированного порядка обхода;
- обращает направление хотя бы одного provenance-edge;
- объединяет atom-codes с разными origin-id или sorts;
- трактует hyperedge как дизъюнкцию premises;

- принимает непомеченное empty-premise rule;
- допускает отрицательное или удаляющее правило как positive monotone;
- позволяет payload создать protected constructor;
- пропускает reachable introduction либо добавляет unreachable atom;
- объявляет весь произвольный fixed point grounded только из равенства $X = T_s(X)$;
- теряет одно из несравнимых minimal bases либо сохраняет заведомо неминимальный superset;
- аварийно завершается на структурно невалидном входе вместо ограниченного статуса.

9.3. Несостоятельность применения

Теорема может оставаться верной, но быть применена неверно, если:

- SCC record-графа выдан за физическую причинную петлю;
- protected atom выдан за референт;
- seed назван ложным только потому, что он не выведен этой цепью;
- circular support назван логическим противоречием без отдельного consistency contract;
- gfp объявлен недопустимым во всех coinductive системах;
- finite checker выдан за доказательство о любой бесконечной рекурсии;
- локально originless SCC назван globally originless при наличии incoming introduction;
- успешное применение, sensor или goal скрыто присутствуют во входе, но объявлены результатом одной mathematical projection;
- отсутствие goal-record расширено до отсутствия цели;
- отсутствие finite support-tree расширено до ложности payload;
- для реальной практики не предъявлен перевод, сохраняющий rules, readset, CP sorts, SG role-codes, origins, seeds и fixed-point policy;
- заголовок расширен до «вся математика ложна» или «никакой расчёт не работает».

10. Обязательные положительные и отрицательные controls

Колонка ниже содержит смешанные уровни только с явной маркировкой: schema-status или reason-code названы как поля; остальные элементы являются проверяемыми observations/protocol-labels, а не значениями верхнеуровневого status.

Control	Ожидаемый record-результат
Один local introduction, ациклический путь к target	status: VALID-CYCLIC-PROVENANCE-LFP; atom присутствует у всех reachable descendants
Originless self-loop	NO-ORIGINLESS-ATOM-IN-LFP; least provenance пусто
Originless двухузловой SCC	NO-ORIGINLESS-ATOM-IN-LFP; непустые equation-solutions помечаются nonleast
SCC с local introduction	CYCLE-PROPAGATES-INTRODUCED
SCC без local introduction, но с incoming origin	CYCLE-PROPAGATES-INTRODUCED
Одинаковый payload при разных origin-id	Два разных protected atoms
Payload содержит полную сериализацию origin/seed	Protected closure не меняется
Tagged executable seeded-cycle: BASE B:seed; CLAIMs C:p, C:q; B:seed $\Rightarrow$ C:p, C:p $\Rightarrow$ C:q, C:q $\Rightarrow$ C:p	status: VALID-SUPPORT-GROUNDING-LFP; для C:p и C:q exact minimal base-set равен {B:seed}
Только C:p $\Rightarrow$ C:q и C:q $\Rightarrow$ C:p, без BASE и без любого альтернативного конечного seed-grounded support-tree	protocol-label NO-SEED-NO-ROOT-IN-LFP; C:p,C:q отсутствуют в μT_S
Inference с пустыми premise_claim_ids и required_base_ids	status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT; reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE]
Нулевой constructor, нормализованный в BASE с exact base_id (например, convention-id RULE-SEED:r0) и явно указанный тем же id в required_base_ids	Head grounded через protected base; prefix RULE-SEED: не требуется checker-ом
Hyperrule $\{p_1, p_2\} \rightarrow q$, доступен только p_1	$q \notin \mu T_S$
Два несравнимых minimal seed-bases	Оба присутствуют в $B_{\min}(q)$
Nonleast fixed point seedless claim-cycle	UNSUPPORTED-NONLEAST-FIXED-POINT
Явно выбранная gfp policy	FIXED-POINT-POLICY-SUPPLIED
Nonmonotone recursion	OUTSIDE-CP-LFP-3-NONMONOTONE / OUTSIDE-SG-LFP-3-NONMONOTONE / OUTSIDE-FP-SEL-3-NONMONOTONE
Максимально успешная rocket-трасса с введёнными bridge/telemetry/criterion	Успех допускается; introduced codes grounded, но не ретроактивны
Система, которая принимает только удобные originless примеры и не принимает цикл с введённым atom, зашивает лозунг вместо проверки теоремы.	

11. Точный реестр claim / status / falsifier

Глобальный status-layer fence из § 0 действует на всю таблицу: только строки после явного status: являются literal верхнеуровневыми enum; reason_codes остаются отдельным полем; ABSENT, CUT-CERTIFIED, UNBLOCKABLE-PROTECTED-ENDPOINT — только literal вложенного cut_status; self-claim status указывается как expected_status; остальные CAPS-строки являются protocol labels. Wildcard-статусов здесь нет.

Claim	Status	Точный falsifier или предел
$P^\mu(v)$ равно множеству origin-tagged introductions на конечных путях $S \rightarrow^* v$	theorem-id CP-LFP-3 (CFP-3); self-claim expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY	CP-PATH-FALSIFIER: triple присутствует без $(s, \tau, a) \in OIntro(s)$ или без пути $S \rightarrow^* v$; либо такой introduced reachable triple отсутствует в принятом $\mu \Phi_G$
Originless SCC не порождает protected atom в lfp	CP-NOSOURCE • NO-ORIGINLESS-ATOM-IN-LFP	CP-PATH-FALSIFIER в специализации CP-NOSOURCE: atom присутствует, хотя во всём incoming cone нет соответствующего introduction
SCC condensation сохраняет least provenance	CP-SCC • SCC-CONDENSATION-PRESERVES-PROVENANCE	CP-SCC-FALSIFIER: exact condensation с сохранёнными introductions меняет root provenance
Цикл с local или incoming introduction переносит atom	CYCLE-PROPAGATES-INTRODUCED	Reachable introduced atom отсутствует в least provenance
Protected-endpoints internal cut удаляет atom у target тогда и только тогда, когда пересекает все inducer-to-target paths	protocol theorem-label CP-CUT; вложенный cut_status равен ровно ABSENT, CUT-CERTIFIED либо UNBLOCKABLE-PROTECTED-ENDPOINT	CP-CUT-FALSIFIER: все пути пересечены, но atom остаётся; либо surviving path есть, а atom исчез — без rewiring и новых introductions
Target, факторизующийся через frozen core и P^μ , не различает записи с равными значениями этих проекций	CP-AST • CYCLIC-AST-SEPARATED	CP-AST-FALSIFIER: равные core/provenance, разные target-codes при сохранённой тотальной факторизации
$q \in \mu T_S$ iff существует конечное seed-grounded support-tree	theorem-id SG-LFP-3 (GST-3); self-claim expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY; finite fixture status: VALID-SUPPORT-GROUNDING-LFP	SG-SUPPORT-FALSIFIER: одна сторона эквивалентности выполнена, а другая нет при том же finite positive contract
Если для head нет ни одного конечного support-tree с листьями в S , то head отсутствует в lfp; чистый seedless positive cycle — частный случай	protocol-label NO-SEED-NO-ROOT-IN-LFP	SG-SUPPORT-FALSIFIER: head входит в μT_S , хотя конечного seed-grounded tree нет. Наличие отдельного grounded derivation выводит head и не является falsifier

Claim	Status	Точный falsifier или предел
Inference без premise_claim_ids и required_base_ids несовместим с executable contract; нормализация создаёт base и явную ссылку на него	status: INCOMPATIBLE-SUPPORT-GROUNDING-CONTRACT; reason_codes: [NULLARY_INFERENCE_REQUIRES_EXPLICIT_BASE] для ненормализованного случая	Empty-premise inference принят без explicit base либо нормализующий base не указан в required_base_ids
$B_{\min}(q)$ содержит все и только subset-minimal seed leaf-sets	MINIMAL-ROOT-BASES-ENUMERATED	SG-BASIS-FALSIFIER: пропущено минимальное основание, добавлен strict superset либо лист не является protected seed
Root исчезает после seed deletion iff disabling set пересекает каждый minimal basis	ROOT-SEED-CUT-VALID	SG-HIT-FALSIFIER: все bases пересечены, но root остаётся, либо surviving basis есть, но root исчезает
Support-grounded fixed point единственен и равен μT_S	theorem-id FP-SEL-3 (FPS-3); self-claim expected_status: TEXTUAL-THEOREM-IN-DECLARED-METATHEORY; protocol-label UNIQUE-SUPPORT-GROUNDED-FIXED-POINT	FP-GROUNDING-FALSIFIER
Каждая fixed point лежит между least и greatest	LEAST-FIXED-POINT-FROM-BOTTOM / GREATEST-FIXED-POINT-FROM-TOP	FP-INTERVAL-FALSIFIER: exact fixed point не содержит lfp либо не содержится в gfp
Nonleast fixed point содержит unsupported residue, ведущий backward к dependency-cycle	UNSUPPORTED-NONLEAST-FIXED-POINT / CIRCULAR-RESIDUE-LOCALIZED	FP-RESIDUE-FALSIFIER; частный grounding-falsifier: $X \neq \mu T_S$, но каждый $q \in X$ имеет конечное seed-tree
Выбор lfp, gfp или иной semantics является полем контракта	FIXED-POINT-POLICY-SUPPLIED / POLICY-SENSITIVE-CODE	FP-POLICY-FALSIFIER; реализационный предел: output меняется от policy, но policy отсутствует в declared readset
Gfp всегда отличается от lfp и всегда добавляет residue	REJECTED-OVERREACH	Любой контракт с $\forall T_S = \mu T_S$
Nonmonotone recursion покрыта CFP-3/GST-3/FPS-3	OUTSIDE-CP-LFP-3-NONMONOTONE / OUTSIDE-SG-LFP-3-NONMONOTONE / OUTSIDE-FP-SEL-3-NONMONOTONE	Для применения требуется отдельная теорема; текущая строка не принимается
Payload способен назначить себе origin, seed, rule или policy	PAYLOAD-SPOOF-REJECTED / PAYLOAD-SUPPORT-SPOOF-REJECTED / PAYLOAD-POLICY-SPOOF-REJECTED	Protected constructors не читаются из payload; реализация, делающая это, поражена
Declared readset является полным только потому, что так написано	NOT-ATTESTED	Два запуска с одинаковым declared input и разным output из-за скрытого channel

Claim	Status	Точный falsifier или предел
Open или бесконечная цепь без seed является finite support-tree	REJECTED	По определению finite tree; бесконечная область требует нового контракта
Максимальный rocket-success опровергает Phase 3	NOT-A-FALSIFIER • ADDED-INTRODUCTIONS	Bridge, telemetry, criterion, mechanism и goal входят как новые protected records
CFP-3 доказывает отсутствие физической причинности	REJECTED-CATEGORY-TRANSFER	Результат относится к path provenance codes, не к physical cause
Отсутствие seed доказывает ложность payload	REJECTED-POLARITY-EXPORT	GST устанавливает только отсутствие finite seed-tree в declared support system
Этот документ исключён из собственной границы	REJECTED-SELF-EXCEPTION	Его формулы и статусы также являются carrier и contract-relative records
Checker-output устанавливает внешний мир	REJECTED-SCOPE-EXPORT	Checker обрабатывает предоставленные байты; отдельного world-relation в контракте нет
Заголовок сам является теоремой	Для зарегистрированного в 00_MANUSCRIPT_RU.md claim C-PUBLIC-TITLE: kind: title; expected_status: UNBOUNDED-STANDALONE. Заголовки этого adversarial-файла отдельно не зарегистрированы	Теоремный статус имеет только точная ограниченная формула CFP-3/GST-3/FPS-3
Phase 3 установила исторический приоритет	novelty_status: NOT-ASSESSED; priority_status: NO-PRIORITY-CLAIM	Исторический claim не является посылкой и требует отдельного контракта

12. Финальная граница красной команды

После всех атак сохраняется точный результат:

В конечном положительном контракте least-provenance содержат ровно origin-tagged introductions, достигающие узла по конечным путям. Цикл может переносить введённое, но originless SCC не вводит protected atom. В support-части nullary constructors предварительно нормализованы в protected bases, а ordinary rules имеют непустые premises: code входит в grounded least fixed point ровно при наличии конечного seed-support tree. Любая $X \in \text{Fix}(T_S)$, отличная от μT_S , содержит residue без finite seed-tree; для unary Φ_G действует отдельное path-grounded утверждение. Выбор иной fixed-point policy остаётся добавленным полем контракта.

Короткая публичная форма, не разрушающая теорему:

Цикл может повторить введённое. Источником повторение не стало.

Даже эта строка остаётся carrier. Она не предъявляет источник, цикл, ракету, полёт, цель или отсутствие смысла.