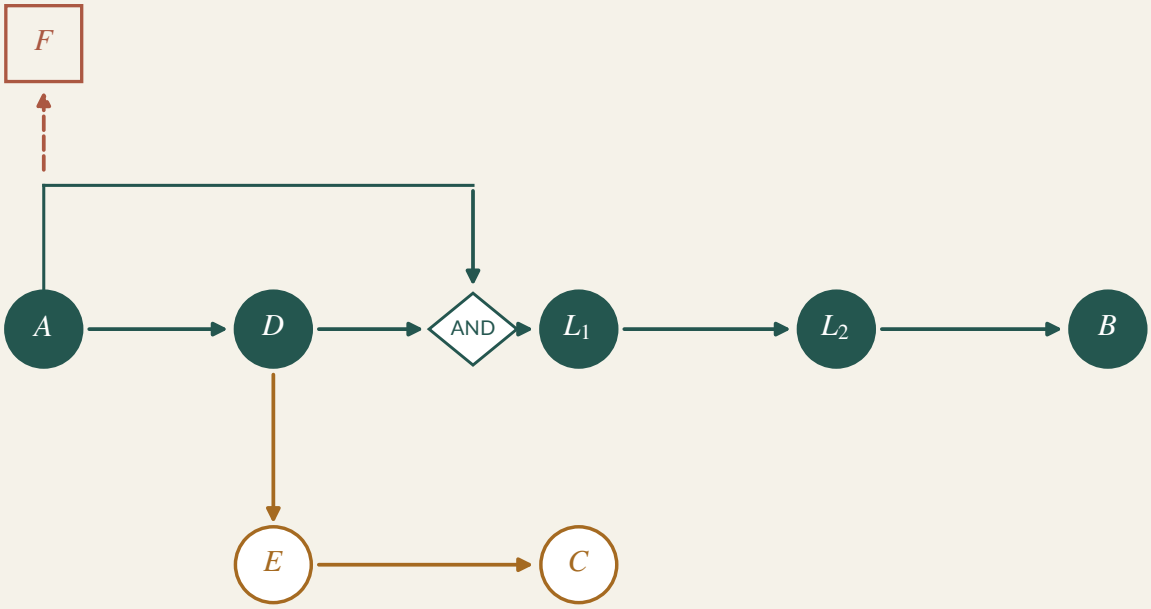
OPERATION KOMOREBI

REALIZATION CALCULUS / MATHEMATICAL REASONING

Proof Path Theory

Certificates, compression, discovery fields, and mathematical memory



The proof is a certified core. The field retains the surrounding mathematics.

Komorebi P.

Monograph edition 1.8.0 / 19 October 2026

Reading map

The five parts follow one mathematical record from discovery to certification, realization, compression, review, and reuse. The visual studies work through actual finite examples rather than replacing the proofs.

01 FIELD
What can be derived?
Certified rules and an explicit record of failure.

02 REALIZATION
Can local descriptions fit together?
Restrictions, whole supports, fibers and witnesses.

03 MEMORY
What survives compression?
One decoder per task; retained and lost distinctions.

04 INQUIRY
What changes during review?
Provenance, shared error, audit capacity and depth.

05 EXECUTION
What can be checked independently?
Finite records, exact examples and scoped formal evidence.

Figure 1. Five questions organize the monograph. The order is a reading route, not a claim that one kind of evidence replaces another.

A short technical route

Begin with Sections 3, 5, 9, and 10. For collective interpretation, read Sections 20 and 21 together.

Evidence in this edition

The 41 prior result groups are retained. Six new groups specify trace checking and typed exchange. Eighteen Lean modules were freshly built, replayed with Lean's kernel, and axiom audited. Exact finite checks remain separate from the formal proofs.

PART II

Coherent realization and information

Compatibility, extension, obstruction, and ambiguity become concrete once the views and protected observations are specified. This part gives the actual bridge to Komorebi Space.

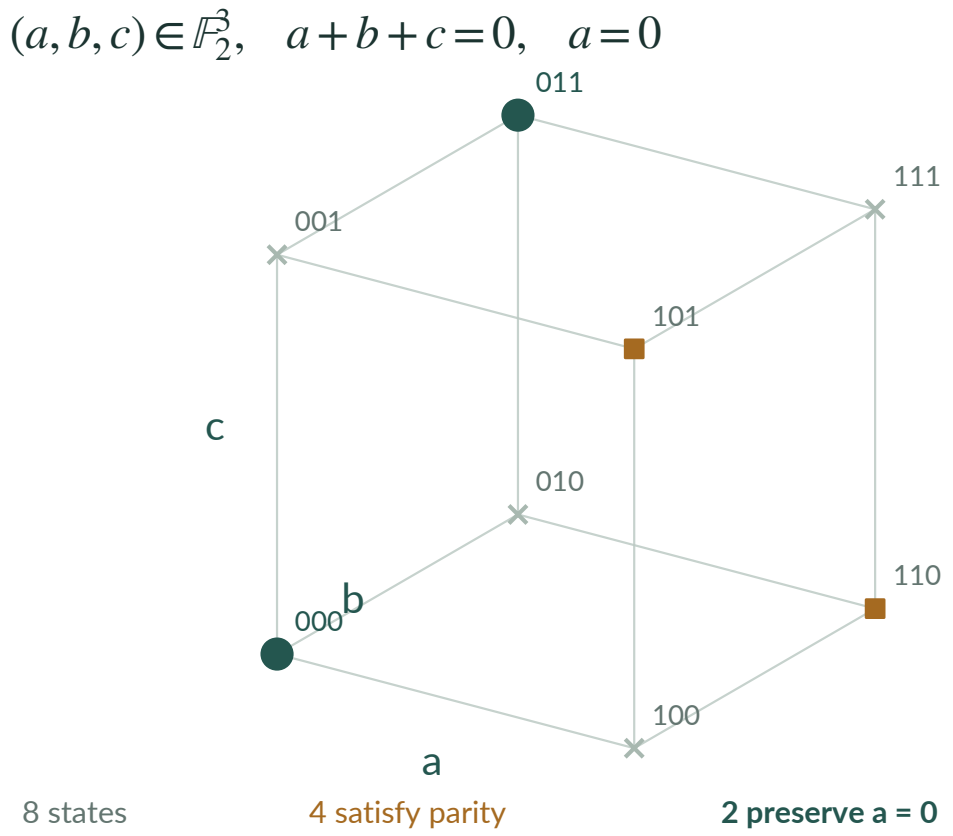


Figure 7. Eight vertices encode the states of $\mathbb{F}_2^3$. Four satisfy $a + b + c = 0$. Protecting $a = 0$ leaves the two green vertices 000 and 011. The cube edges show single-coordinate changes, not continuous solution lines or proof steps.

The governing distinction

Existence asks whether the fiber is empty. Ambiguity asks what remains inside it. An informative observation distinguishes points of that fiber rather than merely producing more output coordinates.

VISUAL STUDY 02

Agreement can hide a missing rule

Keep the same rule and choose the local data $X_0 = \{a\}$ on $U_0 = \{a, c\}$ and $X_1 = \{b\}$ on $U_1 = \{b, c\}$. Both local models assign false to their shared coordinate c. They match exactly, yet their only assembled valuation violates the global rule.

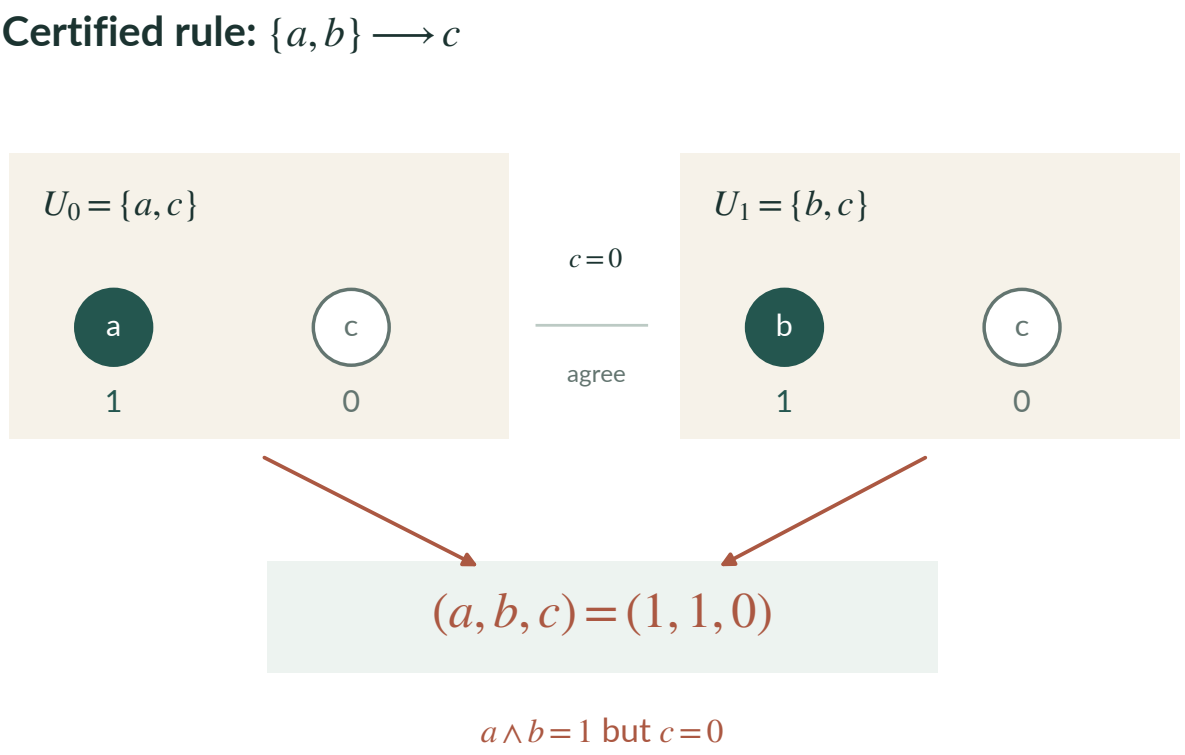


Figure 9. The explicit counterexample to replacing whole-support coverage with pairwise dependency coverage. The local data force $(a, b, c) = (1, 1, 0)$, so both premises hold and the conclusion fails. Green nodes carry value one and open nodes carry value zero.

The successful gluing argument

A patch containing the entire support $\{a, b, c\}$ must enforce the conjunctive rule. If all local models match and every certified support occurs in a patch, their union obeys every rule. Node coverage then gives uniqueness. This is the content of Theorem 9.3, not an inference from pairwise agreement alone.

The cover condition is sufficient for all matching data. It need not be necessary for an individual field whose other rules already make a support constraint redundant.

VISUAL STUDY 05

A missing support can be redundant

The patches $U_0 = \{a, b\}$ and $U_1 = \{b, c\}$ do not contain the support of $a \to c$. Their visible rules $a \to b$ and $b \to c$ nevertheless provide a certified derivation of that implication. Compare this with the missing conjunctive support in Figure 9: neither visible patch there provides a substitute for $\{a, b\} \to c$.

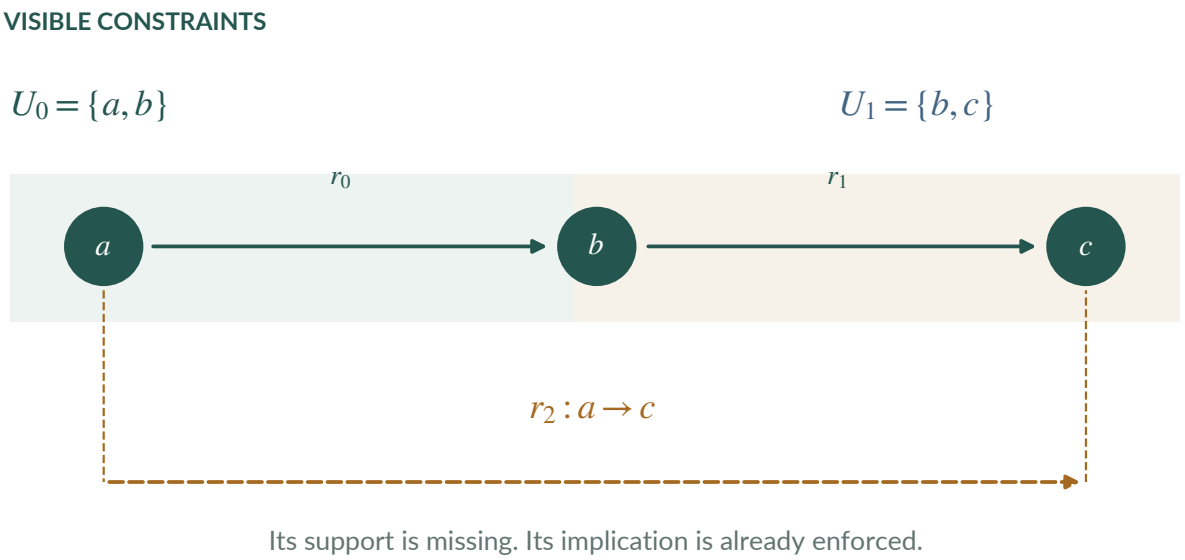


Figure 13. The exact semantic-cover check for a redundant rule. The two solid rules are visible in the displayed patches. The dashed original rule has no covered whole support, but its conclusion is derived by the solid path. Dashed here means omitted from the visible subsystem, not uncertified in the original field.

Preserve the constraint, not necessarily its original presentation

The cover is sufficient because its visible rules impose the same models as the full field. The criterion checks every original rule, not just the target of one successful example.

For a finite field, one closure computation per original rule is enough to produce the positive certificates or expose a failed entailment. This avoids enumerating all local valuations or all matching families. The implementation uses repeated rule scans, so the release does not claim the indexed linear-time bound for its code. Its output certificates can be verified without trusting that search procedure.

9.11 Executable certificates for the declared closure

A mathematical characterization of closure is not yet a correctness theorem for the program that checks a certificate. The next construction closes that boundary for an explicit Lean checker. It uses the existing `Field` and seeded-closure definitions, rather than replacing their meaning by the output of an algorithm.

Definition 9.13 (Finite indexed problem and trace)**.** An indexed problem P specifies a finite address space $V_n = \{0, \dots, n-1\}$, admitted leaves L, a list of certified rules, positive observations S, and forbidden nodes N. A rule at address j has a finite premise list A_j and conclusion b_j. Two identical rows may have different rule addresses. Every listed node must belong to V_n. Node lists are individually duplicate free, but L, S, and N need not be disjoint.

A trace $\sigma = (j_1, \dots, j_k)$ is a list of rule addresses. Replay begins with the list $K_0 = L +\!\!+ S$. It rejects an unknown rule address or a step whose premises are not all known. Otherwise it adds the conclusion. Strict replay additionally requires the conclusion to be new. Causal replay permits a conclusion already known. Lists are compared by membership when they represent sets.

Only rows declared certified enter this indexed rule list. The finite checker does not infer certification from a claim's label, its position in a document, or an author's confidence. An adapter that selects these rows must preserve the intended field and its assumptions.

Theorem 9.14 (Replay preserves derivability and composes)**.** *Suppose every member of K belongs to $\mathrm{cl}_F(S)$. If either replay mode accepts a trace starting from K and returns K', then*

$$K \subseteq K' \subseteq \mathrm{cl}_F(S).$$

For a fixed replay mode, replaying $\sigma +\!\!+ \tau$ is the same as replaying σ and then replaying τ from the returned state. Failure at either stage propagates to the concatenated execution.

Proof. Induct on the trace. The empty trace returns the initial state. For a nonempty trace, a successful first step identifies a certified rule whose premises are already derivable. Closure under that rule proves its conclusion derivable. Adding the conclusion preserves all previously known nodes, and the induction hypothesis applies to the remaining trace. The stricter novelty test can reject a step but cannot invalidate this argument for an accepted one. The composition identity follows by the same induction, with the lookup and enabling tests taking the same branch on both sides. □

The functions `replay`, `enabled`, and `lookup` are executable definitions. The formal theorems `replay_sound`, `replay_extensive`, and `replay_append` concern those definitions themselves. They are not specifications assumed of an otherwise unchecked replay routine.

Theorem 9.15 (Acceptance identifies the least seeded model)**.** *A complete-closure certificate consists of a strict trace σ and a reported node list K. The checker accepts precisely when replay succeeds, its output has the same members as K, and K contains the admitted leaves and satisfies every declared rule. Acceptance implies*

$$\boxed{K = \mathrm{cl}_F(S).}$$

Here equality means equality of the represented sets, not equality of list order.

Proof. Theorem 9.14 gives $K \subseteq \mathrm{cl}_F(S)$ because every initial node is an admitted leaf or a positive observation. Extensivity of replay ensures that K contains S. The final closedness test makes K an F-model, so leastness of seeded closure gives the reverse inclusion. Both tests are needed: replay provides derivability, while the model test provides saturation under rules not used in the trace. □

VISUAL STUDY 06

Why replay and closedness are both necessary

Take $V_4 = \{0, 1, 2, 3\}$, $L = \{0\}$, $S = \varnothing$, and the two rules $r_0 : 0 \to 1$ and $r_1 : 1 \to 2$. The least closure is $\{0, 1, 2\}$. Node 3 has no derivation from these inputs.

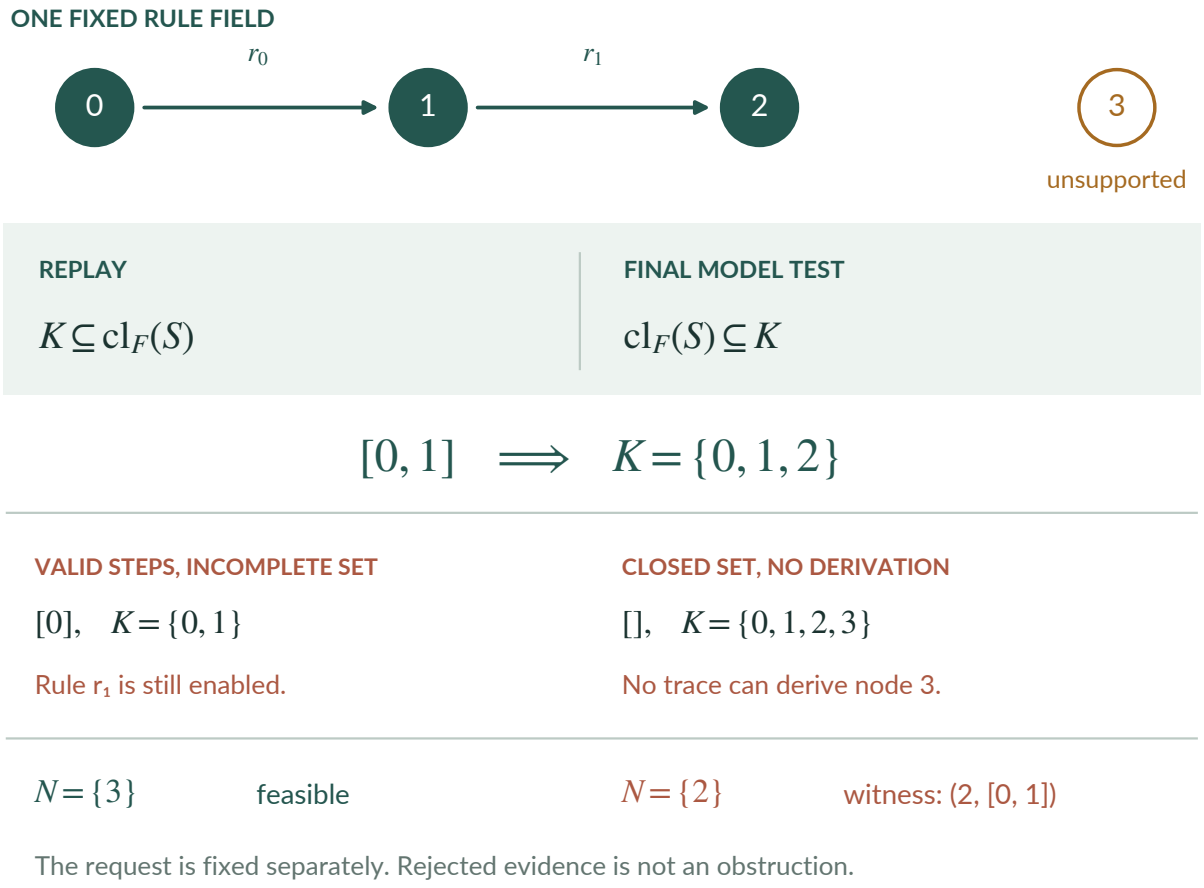


Figure 14. Replay and the final model test establish opposite inclusions. Together they identify the least closure. The two rejected records fail different obligations. Changing only the forbidden set changes the correct feasibility verdict without changing the closure.

The trace $[0]$ derives exactly $\{0, 1\}$ but leaves r_1 enabled with its conclusion absent. It is a valid partial derivation, not a complete closure. Conversely, $\{0, 1, 2, 3\}$ is closed under both rules, but replay cannot establish its unsupported node 3. A model need not be the least model.

With $N = \{3\}$, the exact closure gives a positive result. With $N = \{2\}$, the same closure supports a negative result with witness $(2, [0, 1])$. The second problem is not a different interpretation of the same request: its protected observations have changed. A checker must receive the fixed request separately from the proposed answer.

Corollary 9.18 (Meaning of a complete result certificate)**.** *The complete result format contains a Boolean feasibility flag, a reported closure, a strict closure trace, and an optional forbidden-node witness. Both branches require the complete-closure check. A positive result additionally requires no obstruction witness and no forbidden node in the closure. A negative result requires an accepted obstruction witness. For every accepted result C,*

$$C.K = \mathrm{cl}_F(S), \qquad \big(C.\text{feasible} = \text{true}\big) \iff \mathrm{Ext}_F(S, N) \neq \varnothing.$$

Proof. The complete-closure check proves the first equality. In the positive branch, the closure is a model containing S and avoiding N, hence is an extension. In the negative branch, the witness excludes every extension by Theorem 9.17. Case analysis on the Boolean flag gives the equivalence. □

A bare negative witness has its own checker and does not need a saturated closure. The complete result format asks for both because it promises both an exact closure and a feasibility verdict. Failure of either check means that this certificate was not accepted. It is not itself a refutation of the protected problem.

9.13 Typed exchange and the boundary of serialization

A certificate is useful outside its producer only when the receiving system knows exactly which data it checked. The exchange format therefore uses versioned positional arrays, exact nonnegative integer tokens, literal Booleans, and explicit rule addresses. Problem and certificate are separate records. The node address book and the selected certified-rule order remain associated with the originating proof-field snapshot.

At the JSON syntax-tree level, the problem has the shape

```
["proof-path/problem/1", node_count, leaves, rules, positive, negative]
```

where each rule is `[premises, conclusion]`. The certificate has the shape

```
["proof-path/certificate/1", feasible, closure, trace, obstruction]
```

where an obstruction is either `null` or `[target, trace]`. Array lengths and tags are exact. A Boolean is not an integer address, and the string `"false"` is not a Boolean verdict. Duplicate node entries are rejected by validity checks, while identical rule rows at different addresses remain distinguishable.

Theorem 9.19 (Lossless encoding at the typed syntax boundary)**.** *Let e_P, d_P and e_C, d_C be the stated Lean encoders and partial decoders for problem and certificate records. For every typed problem P and certificate C,*

$$d_P(e_P(P)) = \mathrm{some}(P), \qquad d_C(e_C(C)) = \mathrm{some}(C).$$

Consequently both encoders are injective on their typed domains. These identities do not require the encoded record to pass the separate validity checks.

Proof. The natural-number and Boolean constructors decode to their original values. Induction on a list proves the corresponding array round trip. The rule, witness, and optional-witness decoders then recover their components. Unfolding the exact-length record encoders and decoders gives the two identities. If two encoded records are equal, applying the decoder gives equal `some` values, hence equal original records. □

Theorem 9.20 (Accepted JSON records certify their decoded answer)**.** *Suppose the JSON checker accepts the separately supplied problem and certificate syntax trees. Then both decode to typed records P, C, both pass their index and shape checks, and the decoded certificate satisfies the closure equality and feasibility equivalence of Corollary 9.18 for that decoded problem.*