Boundary Discipline and the Structural Limits of Internal Certification

Duston Moore

Abstract Verification pipelines across logic, software, and artificial intelligence systematically separate generative components from certification mechanisms. This separation is often treated as a contingent engineering choice. This paper argues instead that it reflects a structural limit on internal certification.

We develop three complementary results. First, we analyse verification architectures in terms of boundary transitions between layers and show that the most severe correctness failures arise at these boundaries rather than within individual components. Second, we introduce a formal framework of boundary-constrained defeasible consequence and prove that no sentential calculus satisfying standard extensional replacement principles can be sound for such consequence. Third, we establish a production–closure separation theorem for constructive systems, showing that no closure operation can be internally realised as a finite composition of productive operations.

We further show that probabilistic methods do not evade these limits. A Measure–Partition Mismatch theorem demonstrates that probabilistic updating cannot correct ontological misalignment when safety-critical distinctions are absent from the event algebra. Finally, we extract architectural consequences for verification-aware AI systems and formalise an explicit sealing protocol that enforces production–closure separation.

Together, these results show that systems capable of indefinite productive extension cannot internally certify the closure conditions that stabilise their own outputs. Boundary discipline is therefore not optional but a necessary structural principle for reliable verification.

Keywords: Formal verification · Boundary errors · Defeasible consequence · Hyperintensional logic · AI safety · Certification architectures

1 Introduction

Formal verification systems are almost universally structured as layered architectures. Generative components propose candidates, while distinct certification mechanisms check, validate, or stabilise those candidates relative to a specification. This pattern is ubiquitous and appears in proof assistants, model checking pipelines, static analysis frameworks, and increasingly in the verification of machine-learning systems. This separation is often treated as a pragmatic design decision motivated by engineering convenience, performance, or trust management.

The central claim of this paper is that the separation between generation and certification reflects a structural limit on what can be internally certified by productive systems.

The problem addressed here is not the familiar issue of incorrect outputs produced by heuristic components. Errors internal to a layer are typically detectable by downstream checks. Rather, the most severe verification failures occur at the boundaries between layers, where artefacts are translated, embedded, or interpreted across differing admissibility regimes. These failures are not attributable to bugs in any single component. They arise from structural mismatches between generative and certifying contexts.

This paper develops a formal account of such boundary failures and establishes precise limits on internal certification. The guiding intuition is simple but far-reaching: productive operations preserve openness, while certification requires closure. No finite composition of productive operations can, in general, achieve closure without collapsing distinctions essential to correctness.

We substantiate this intuition through three lines of argument.

First, we analyse verification architectures in terms of boundary transitions and show that correctness guarantees are not compositional across boundaries. Verification success within a layer does not entail correctness relative to the surrounding system unless boundary conditions are explicitly enforced.

Second, we formalise boundary-constrained defeasible consequence. This framework captures the requirement that admissible inference must respect the case structure induced by premises. We prove a negative result showing that no sentential calculus satisfying standard extensional replacement principles can be sound for such consequence. Classical extensionality collapses distinctions that boundary-sensitive inference must preserve.

Third, we examine constructive systems and proof assistants. We prove a production–closure separation theorem showing that no closure operation deciding proof completeness can be internally expressed as a finite composition of productive transformations. This result explains the architectural necessity of kernel–tactic separation and limits the scope of reflective techniques.

A natural response to these limitations is to appeal to probabilistic reasoning. We show that this move fails in principle. Probabilistic methods presuppose an event algebra that already encodes the distinctions relevant to certification. When safety-critical distinctions fall outside this algebra, probabilistic updating converges to high-confidence error rather than correction.

The contributions of this paper are therefore both formal and architectural. Formally, we establish impossibility results that constrain the design of verification systems. Architecturally, we extract a boundary discipline principle that mandates explicit closure mechanisms external to productive processes.

The remainder of the paper is organised as follows. Section 2 introduces a formal model of verification boundaries and classifies boundary errors. Section 3 develops boundary-constrained defeasible consequence and proves the incompatibility with extensional calculi. Section 5 establishes the probabilistic limitation result. Section 6 proves the production–closure separation theorem. Section **??**

derives architectural consequences for verification-aware AI systems. Section **??** concludes.

2 Boundary Errors and Verification Architecture

Formal verification systems are rarely monolithic. Instead, they are organised as pipelines in which artefacts are produced, transformed, and certified under progressively stricter admissibility conditions. Correctness guarantees are typically local to individual stages of such pipelines. The claim of this section is that the most severe verification failures arise not within stages, but at the boundaries between them.

To make this precise, we introduce a formal model of verification pipelines and define boundary errors as structural failures of translation or interpretation across stages.

2.1 Verification Pipelines

A verification pipeline consists of a finite sequence of components

$$C_0 \xrightarrow{\tau_0} C_1 \xrightarrow{\tau_1} \dots \xrightarrow{\tau_{n-1}} C_n$$

where each component C_i operates over a domain of artefacts $\mathcal{A}_i$ equipped with an admissibility predicate

$$\mathsf{Adm}_i : \mathcal{A}_i \rightharpoonup \{\top, \bot\}.$$

The interpretation of admissibility differs across components. Early stages typically admit artefacts heuristically or provisionally, while later stages enforce stricter correctness criteria.

Each transition $\tau_i : \mathcal{A}_i \to \mathcal{A}_{i+1}$ represents a boundary at which artefacts are translated, embedded, or reinterpreted. Verification guarantees do not automatically propagate across these transitions.

Definition 1 (Pipeline Correctness). *A pipeline is locally correct if, for each component C_i, all artefacts accepted by Adm_i satisfy the specification internal to C_i.*

A pipeline is globally correct if acceptance at C_n entails satisfaction of the intended system-level property.

Local correctness does not imply global correctness. Boundary errors account for this gap.

2.2 Boundary Errors

We define a boundary error as a failure of correctness preservation across a pipeline transition.

Definition 2 (Internal and External Specifications). *Each component C_i is associated with:*

1. *an* internal specification Φ_i^{int} *defining the property checked by* Adm_i*, and*
2. *an* external specification Φ_i^{ext} *defining the intended system-level invariant that C_i is meant to contribute to preserving.*

The internal specification is formal and local. The external specification encodes environmental or deployment assumptions and need not be directly checkable by C_i.

Definition 3 (Boundary Error). *A boundary error occurs at transition τ_i if there exists an artefact $a \in \mathcal{A}_i$ such that:*

1. $\mathsf{Adm}_{i+1}(\tau_i(a)) = \top$,
2. $\tau_i(a)$ *satisfies* $\Phi_{i+1}^{\mathsf{int}}$*, and*
3. τ_i *fails to preserve satisfaction of* $\Phi_{i+1}^{\mathsf{ext}}$.

Boundary errors are not detectable by examining any single component in isolation. Each component behaves correctly relative to its own specification.

We distinguish two structurally distinct classes of boundary error.

2.3 Embedding Boundary Errors

An embedding boundary error arises when the artefact presented to a certifying component does not faithfully represent the artefact generated upstream.

Definition 4 (Embedding Boundary). *An embedding boundary is a transition τ_i such that correctness requires preservation of semantic structure:*

$$[\![a]\!]_i = [\![\tau_i(a)]\!]_{i+1}.$$

An embedding boundary error occurs when this condition fails.

Examples include incorrect translation of proof obligations, lossy compilation of specifications, or mismatched type encodings between generators and checkers.

2.4 Interpretation Boundary Errors

An interpretation boundary error arises when the property certified by a verifier does not entail the intended system-level guarantee.

Definition 5 (Interpretation Boundary). *An interpretation boundary is a transition at which verified properties must be mapped to external meaning. An interpretation boundary error occurs when*

$$\mathsf{Adm}_{i+1}(\tau_i(a)) = \top \quad \textit{but} \quad [\![\tau_i(a)]\!]_{i+1} \not\models \Phi_{i+1}^{\mathsf{ext}}.$$

Interpretation boundaries are common in verification of deployed systems, where verified models abstract away environmental or representational assumptions.

2.5 Verification Layers as Boundary Structure

For concreteness, many verification pipelines can be abstracted into the following stages:

- **Generative exploration:** heuristic production of candidate artefacts
- **Grounding and translation:** embedding into formal representations
- **Structural validity:** syntactic and type correctness
- **Semantic verification:** proof, model checking, or constraint solving
- **System-level enforcement:** runtime monitoring or deployment guarantees

Each adjacent pair induces a boundary in the sense defined above. Boundary errors are therefore intrinsic to layered verification, not exceptional anomalies.

2.6 Case Study: Scope Mismatch in Smart Contract Verification

We illustrate an interpretation boundary error using a simplified smart contract verification pipeline.

1. A safety property is specified informally as "account balances never become negative".
2. The specification is formalised as an invariant over unsigned 256-bit integers.
3. An SMT solver proves the invariant holds for all reachable states.
4. The contract is deployed in an execution environment using signed 64-bit integers.

Each component satisfies its local correctness condition. The solver proves the invariant relative to the formal model. Deployment respects the compiled contract semantics.

Nevertheless, the intended safety property fails. The interpretation boundary between the verified model and the deployment environment is open. Acceptance by the verifier does not entail the intended external guarantee.

This failure is invisible to any single verification stage. It is a boundary error in the sense defined above.

2.7 Consequences

Boundary errors demonstrate that verification correctness is not compositional across pipeline stages. Correctness guarantees must be explicitly scoped and enforced at boundaries.

This observation motivates the formal development in the next section, where we introduce a logic of boundary-constrained defeasible consequence capable of expressing admissibility relative to explicit case partitions.

3 Boundary-Constrained Defeasible Consequence

Boundary errors arise when admissibility conditions governing verification are silently altered across pipeline transitions. To reason about such phenomena formally, we require a notion of consequence that is sensitive not only to truth preservation but also to the case structure induced by premises and boundary constraints.

This section introduces boundary-constrained defeasible consequence. The framework captures the requirement that inference be admissible only relative to explicitly represented cases. The resulting notion of consequence is hyperintensional and departs from classical extensional reasoning.

3.1 Regions and Admissibility

Let $\mathcal{L}$ be a propositional language and let $\mathcal{W}$ be a space of semantic possibilities. We write $w \models \phi$ for satisfaction in the usual sense.

Definition 6 (Region). *A region is a set $R \subseteq \mathcal{W}$ representing the possibilities that remain admissible relative to a given verification context.*

Verification contexts induce regions by excluding possibilities that violate established constraints.

Definition 7 (Live Region Operator). *A live region operator is a function*

$$\mathsf{Reg} : \mathcal{P}_{\mathrm{fin}}(\mathcal{L}) \to \mathcal{P}(\mathcal{W})$$

satisfying the shrinkage condition:

$$\mathsf{Reg}(\Gamma \cup \{\psi\}) \subseteq \mathsf{Reg}(\Gamma).$$

Intuitively, $\mathsf{Reg}(\Gamma)$ represents the possibilities consistent with the premises Γ under the verification scope currently in force.

3.2 Case Partitions and Boundary Discipline

Correctness at a boundary requires not merely truth preservation but preservation of admissible cases.

Definition 8 (Case Partition). *Given premises Γ, a family of cases $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a partition of $\mathsf{Reg}(\Gamma)$ if:*

$$\mathsf{Reg}(\Gamma) \subseteq \bigcup_i C_i \quad \textit{(coverage)} \tag{1}$$

$$C_i \cap C_j = \emptyset \quad \textit{for } i \neq j \quad \textit{(disjointness)} \tag{2}$$

$$C_i \subseteq \mathsf{Reg}(\Gamma) \quad \textit{(in-region)}. \tag{3}$$

We refer to the conjunction of these conditions as boundary discipline. No gap ensures that all admissible possibilities are accounted for. No overlap ensures that cases are not silently conflated.

3.3 Mention Sensitivity and Information Dependency

Boundary-constrained reasoning must distinguish between formulae that merely evaluate to the same truth value and formulae that depend on different underlying cases. To capture this distinction, mention sensitivity is grounded in dependency, not in surface presentation.

Definition 9 (Variation Set). *For $\phi \in \mathcal{L}$, define the variation set*

$$\mathsf{Var}(\phi) \;=\; \{\, w \in \mathcal{W} \mid \exists w' \in \mathcal{W} \text{ with } w \models \phi,\; w' \not\models \phi,\; \text{and } w \equiv_\phi w' \,\}.$$

Here $w \equiv_\phi w'$ means that w and w' agree on all atomic evaluations except possibly those relevant to the evaluation of some subformula of ϕ.

Definition 10 (Mentioned Worlds). *Let $\mathsf{Mentioned}(\phi)$ be any operator satisfying:*

$$\mathsf{Var}(\phi) \subseteq \mathsf{Mentioned}(\phi) \subseteq \{w \in \mathcal{W} \mid w \models \phi \text{ or } w \not\models \phi\}.$$

Intuitively, $\mathsf{Mentioned}(\phi)$ must contain all worlds across which the truth of ϕ can vary under admissible refinement.

Condition 1 (Mention Inclusion)**.** *For all $\phi \in \mathcal{L}$,*

$$\mathsf{Mentioned}(\phi) \subseteq \mathsf{Reg}(\{\phi\}).$$

Mention Inclusion requires that any case on which the evaluation of ϕ depends must remain admissible when ϕ is asserted. Boundary discipline therefore forbids conclusions whose evaluation presupposes excluded cases.

3.4 Defeasible Consequence

We now define boundary-constrained defeasible consequence.

Definition 11 (Boundary-Constrained Consequence). *Let $\Gamma \subseteq_{\text{fin}} \mathcal{L}$ and $\psi \in \mathcal{L}$. We write*

$$\Gamma \vdash_{\mathsf{def}} \psi$$

if and only if:

1. $\mathsf{Reg}(\{\psi\}) \subseteq \mathsf{Reg}(\Gamma)$, *and*
2. *for every case partition $\mathcal{C}$ of $\mathsf{Reg}(\Gamma)$, ψ is true throughout each $C_i \in \mathcal{C}$.*

The first condition enforces boundary admissibility. The second enforces case-respecting necessity. Failure of either condition blocks inference.

3.5 Hyperintensionality

Boundary-constrained consequence is hyperintensional. Logically equivalent formulae may differ in admissibility due to boundary conditions.

Example 1. Let $\chi = A \vee B$ and $\psi = (A \vee B) \vee (A \wedge B)$. These formulae are classically equivalent. However, ψ explicitly involves the joint case $A \wedge B$ in a way that can force admissibility commitments that χ does not.

In particular, if boundary constraints exclude the joint case from $\mathsf{Reg}(\Gamma)$, then $\Gamma \nvdash_{\mathsf{def}} \psi$ even when $\Gamma \vdash_{\mathsf{def}} \chi$.

3.6 Relevance Quotients

Strict mention sensitivity can be overly fine-grained in practice. Verification engineers often intentionally ignore distinctions deemed irrelevant to a given scope.

We therefore introduce a controlled relaxation.

Definition 12 (Relevance Congruence). *A relevance congruence $\sim_{\mathcal{R}}$ is an equivalence relation on $\mathcal{L}$ satisfying:*

1. *If $\phi \sim_{\mathcal{R}} \psi$, then $\phi \leftrightarrow \psi$ is a tautology.*
2. *If $\phi \sim_{\mathcal{R}} \psi$, then $\mathsf{Reg}(\{\phi\}) = \mathsf{Reg}(\{\psi\})$.*

Quotienting by $\sim_{\mathcal{R}}$ permits harmless boundary collapses while preserving formal discipline. We return to this relaxation when discussing architectural enforcement mechanisms.

3.7 Outlook

Boundary-constrained defeasible consequence captures the inferential discipline required at verification boundaries. In the next section, we show that this discipline is incompatible with standard extensional calculi, establishing a negative result that underpins the architectural claims of the paper.

4 Incompatibility with Extensional Consequence

The purpose of this section is to establish a negative result. We show that no sentential consequence relation satisfying standard extensional replacement principles can be boundary-sound relative to the admissibility discipline developed above.

4.1 Extensional Consequence Relations

Let $\mathcal{L}$ be a propositional language. We consider consequence relations $\vdash$ satisfying the following conditions.

1. **Soundness.** If $\Gamma \vdash \phi$, then $\Gamma \models \phi$.
2. **Tautological Completeness.** If $\models \phi$, then $\vdash \phi$.
3. **Replacement of Equivalents.** If $\models \phi \leftrightarrow \psi$, then for all Γ,

$$\Gamma \vdash \phi \;\Rightarrow\; \Gamma \vdash \psi.$$

These conditions characterise standard extensional sentential calculi. Replacement is the critical principle. It expresses the idea that logically equivalent formulae are interchangeable in all contexts.

4.2 Boundary-Constrained Soundness

We say that a consequence relation $\vdash$ is boundary-sound if it never derives a formula that violates boundary admissibility.

Definition 13 (Boundary Soundness). *A consequence relation $\vdash$ is boundary-sound with respect to a live region operator* Reg *if, whenever $\Gamma \vdash \phi$, it follows that*

$$\mathsf{Reg}(\{\phi\}) \subseteq \mathsf{Reg}(\Gamma).$$

Boundary soundness requires that derived conclusions do not reintroduce excluded cases.

4.3 The Negative Theorem

Theorem 1 (Extensional Incompatibility). *Let $\vdash$ be a consequence relation satisfying Soundness, Tautological Completeness, and Replacement of Equivalents. Let* Reg *satisfy the shrinkage condition and Mention Inclusion. Then $\vdash$ is not boundary-sound with respect to* Reg*.*

4.4 Proof

Proof. We construct a concrete counterexample.

Worlds. Let $\mathcal{W} = \{w_A, w_B, w_{AB}\}$ where

$$w_A \models A \wedge \neg B, \quad w_B \models \neg A \wedge B, \quad w_{AB} \models A \wedge B.$$

Region. Define a live region operator Reg such that

$$\mathsf{Reg}(\Gamma) = \{w_A, w_B\}.$$

Intuitively, the joint case $A \wedge B$ has been excluded by upstream boundary constraints.

Formulae. Let

$$\chi = A \vee B \quad \text{and} \quad \psi = (A \vee B) \vee (A \wedge B).$$

Classically, χ and ψ are logically equivalent.

Admissibility. By construction,

$$\mathsf{Reg}(\{\chi\}) = \{w_A, w_B\} \subseteq \mathsf{Reg}(\Gamma),$$

so χ is boundary admissible.

However, ψ introduces the joint case $A \wedge B$ in a way that can force admissibility commitments not required by χ. Under Mention Inclusion, this yields:

$$w_{AB} \in \mathsf{Reg}(\{\psi\}).$$

Since $w_{AB} \notin \mathsf{Reg}(\Gamma)$, boundary admissibility fails:

$$\mathsf{Reg}(\{\psi\}) \not\subseteq \mathsf{Reg}(\Gamma).$$

Derivation. By Tautological Completeness, $\vdash (\chi \leftrightarrow \psi)$. By Replacement of Equivalents, if $\Gamma \vdash \chi$, then $\Gamma \vdash \psi$.

Thus $\vdash$ derives ψ from Γ despite ψ violating boundary admissibility.

Conclusion. Therefore $\vdash$ is not boundary-sound.

Alternative Counterexample (Vacuity). Let C be a case excluded by boundary constraints, so that

$$\mathsf{Reg}(\Gamma) \cap C = \emptyset.$$

Let $\phi = C \to \bot$. Classically, ϕ is true throughout $\mathsf{Reg}(\Gamma)$, since C never occurs there.

By Soundness and Tautological Completeness, $\vdash \phi$.

However, verification of ϕ can require access to the excluded case C in order to discharge the implication. Under Mention Inclusion, this forces:

$$\mathsf{Reg}(\{\phi\}) \supseteq C.$$

Since $C \not\subseteq \mathsf{Reg}(\Gamma)$, ϕ violates boundary admissibility.

This example shows that the incompatibility is not an artefact of disjunctive expansion. It arises from the interaction between extensional validity and boundary admissibility itself, and mirrors practical failures where unreachable branches nevertheless generate verification obligations outside the intended scope.

4.5 Interpretation

The theorem shows that extensional replacement forces the derivation of conclusions that reintroduce excluded cases. This is not a peculiarity of the example. Any consequence relation that permits unrestricted replacement of logically equivalent formulae will collapse case distinctions essential to boundary discipline.

The result does not depend on non-classical semantics, probabilistic reasoning, or resource sensitivity. It follows from the interaction between extensionality and the admissibility discipline encoded by Reg and Mention Inclusion.

4.6 Implications for Verification

Verification pipelines routinely exclude cases through abstraction, modelling assumptions, or environmental constraints. Conclusions that reintroduce excluded cases undermine the meaning of certification, even when they are classically valid.

The incompatibility theorem explains why verification frameworks typically restrict logical expressivity, track case structure explicitly, or enforce syntactic discipline at boundaries. It also motivates the architectural separation between productive generation and certifying closure.

In the next section, we show that probabilistic methods do not circumvent this incompatibility but reproduce it in measure-theoretic form.

5 Probabilistic Methods and the Ontology Constraint

It is natural to attempt to repair boundary failures by replacing binary admissibility with probabilistic ranking. This section shows that probabilistic methods cannot resolve boundary misalignment. The limitation is structural and persists independently of data volume, model capacity, or update rule.

5.1 Probabilistic Inference over Fixed Event Algebras

Let $(\Omega, \mathcal{F}, P)$ be a probability space. In probabilistic verification settings, Ω represents a space of behaviours or hypotheses, $\mathcal{F}$ a σ-algebra of events, and P a probability measure.

Bayesian updating modifies P by conditioning on evidence. Such updates redistribute probability mass over events in $\mathcal{F}$. They do not refine $\mathcal{F}$ itself. In particular, probabilistic inference cannot introduce new events or sharpen the partition structure of Ω.

Thus probabilistic inference performs weighting over a fixed ontology.

5.2 Boundary Partitions and Certification

Certification requires that admissible possibilities be partitioned according to boundary-relevant cases.

Let $\Pi = \{C_1, \ldots, C_n\}$ be a boundary partition satisfying no-gap and no-overlap conditions relative to the admissible region. Each C_i represents a case whose distinction is required to assert correctness.

If a case C_i is not representable as an element of $\mathcal{F}$, then no probabilistic method operating on $(\Omega, \mathcal{F}, P)$ can reason about C_i as such. At best, it can approximate C_i by measurable events that fail to coincide with it.

5.3 Measure–Partition Mismatch

Theorem 2 (Measure–Partition Mismatch). *Let $(\Omega, \mathcal{F}, P)$ be a probability space and let $\Pi = \{C_1, \ldots, C_n\}$ be a boundary partition required for certification. If there exists $C_k \in \Pi$ such that $C_k \notin \mathcal{F}$, then no sequence of probabilistic updates on P can yield a certifier that distinguishes C_k with certainty.*

Proof. Since $C_k \notin \mathcal{F}$, there is no measurable event equal to C_k. Any probabilistic procedure must therefore substitute some $E \in \mathcal{F}$ as a proxy for C_k.

Let $\Delta = E \triangle C_k$ be the symmetric difference. Because $E \neq C_k$, Δ is non-empty. The probability space cannot distinguish elements of $C_k \setminus E$ from elements of $E \setminus C_k$.

Bayesian updating may drive $P(E)$ arbitrarily close to 1. However, convergence of $P(E)$ does not eliminate Δ. The distinction required to certify C_k is not expressible in $\mathcal{F}$ and therefore cannot be recovered by probabilistic means.

5.4 Probabilistic Blindness

The preceding theorem shows that probabilistic inference can converge to maximal confidence while remaining insensitive to boundary-critical distinctions. This failure is not epistemic but ontological. The event algebra lacks the resolution required for certification.

We refer to this phenomenon as probabilistic blindness.

5.5 Architectural Consequences

Probabilistic components may guide search, prioritise candidates, or rank hypotheses. They cannot supply certification guarantees unless the boundary partition required for correctness is already represented in their event algebra.

Accordingly, probabilistic generation must remain separated from closure. Certification requires explicit boundary representation and enforcement external to probabilistic inference.

6 Production–Closure Separation

This section establishes that closure cannot be internalised as a finite composition of productive operations. The result is structural and does not depend on a particular proof assistant, logic, or programming language. It explains the architectural necessity of separating generative mechanisms from certifying kernels.

6.1 States, Production, and Closure

Let S be a space of states. In verification contexts, a state may represent a proof obligation, a symbolic execution configuration, or a partially verified artefact.

Definition 14 (Productive Operation). *A productive operation is a (possibly partial) function*

$$f : S \rightharpoonup S$$

that transforms a state into a refined or extended state without guaranteeing termination or completeness.

Productive operations preserve openness. They may generate successors indefinitely.

Definition 15 (Closure Operation). *A closure operation is a predicate*

$$C : S \to \{\top, \bot\}$$

such that $C(s) = \top$ indicates that s satisfies a completeness or finality condition relative to a fixed specification.

Closure determines when productive exploration may soundly stop.

6.2 Internalisation Hypothesis

A common aspiration is to internalise closure by embedding it into productive exploration. Formally, this amounts to the following hypothesis.

Definition 16 (Internalisation Hypothesis). *Closure is internalisable if there exists a finite sequence of productive operations*

$$F = f_n \circ \cdots \circ f_1$$

such that for all states $s \in S$*,*

$$C(s) = \top \quad \textit{if and only if} \quad C(F(s)) = \top.$$

Under this hypothesis, closure is reducible to productive transformation.

6.3 Separation Theorem

Theorem 3 (Production–Closure Separation). *There exist state spaces* S *and closure predicates* C *such that no finite composition of productive operations internalises* C*.*

Proof. Let S be the space of proof states for a sufficiently expressive deductive system. Let $C(s) = \top$ if and only if s represents a complete proof of a fixed proposition.

Assume, for contradiction, that closure is internalisable. Then there exists a finite composition of productive operations F such that $C(F(s)) = \top$ exactly when s is completable.

Since F is computable from productive operations, this yields a procedure that decides, for arbitrary s, whether s can be completed to a proof. This constitutes a decision procedure for proof completeness in the system.

For sufficiently expressive systems, proof completeness is undecidable. Hence no such F can exist.

Therefore closure cannot be internalised as a finite composition of productive operations.

6.4 Scope and Generality

The theorem does not depend on the particular representation of states or operations. Any setting in which productive exploration ranges over an undecidable space admits the separation.

The result applies equally to:

- proof assistants with tactic layers,
- symbolic execution engines,
- constraint solvers with search heuristics,
- generative AI systems producing candidate artefacts.

6.5 Reflection and Partial Internalisation

Reflective techniques allow limited reasoning about states within the system. However, reflective procedures are themselves productive operations. Their outputs must still be validated by an external closure predicate.

Reflection therefore enlarges the productive layer but does not eliminate the need for closure. The separation persists.

6.6 Architectural Consequence

The separation theorem explains the necessity of architectures that isolate closure mechanisms. Trusted kernels, external checkers, and sealing procedures are not engineering conveniences but responses to a structural constraint.

Any system that permits indefinite productive extension must rely on a closure operation that is not reducible to productive exploration itself.