# Boundary Discipline and the Structural Limits of Internal Certification

Duston Moore

**Abstract**

Verification pipelines across logic, software, and artificial intelligence separate generative components from certification mechanisms. This separation is often treated as a contingent engineering choice. The present paper argues that it reflects a structural limit on internal certification. We develop three results. First, we analyse verification architectures in terms of boundary transitions between layers and show that correctness failures concentrate at these boundaries rather than within individual components. Second, we introduce a 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 in sufficiently expressive settings 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 derive architectural consequences for verification-aware systems and formalise an explicit sealing protocol that enforces production-closure separation.

## 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 appears in proof assistants, model checking pipelines, static analysis frameworks, and increasingly in the verification of machine-learning systems. The 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 claim is not that separation is always necessary in every domain, but rather that for systems of sufficient expressive power, internal certification is impossible.

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 verification failures of interest here 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 straightforward: 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. The qualification

"in general" is important. For decidable fragments, internalisation may be possible. The limits we establish concern expressively rich systems.

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 that no sentential calculus satisfying standard extensional replacement principles can be sound for such consequence. The proof proceeds by exhibiting cases where verification of a formula requires reasoning about possibilities that boundary constraints have excluded.

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 in systems where such completeness is undecidable. This result explains the architectural necessity of kernel-tactic separation but does not apply to systems operating in decidable fragments.

A natural response to these limitations is to appeal to probabilistic reasoning. We show that this move fails when the relevant distinctions are not represented in the probability space. 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 confident error rather than correction.

The contributions of this paper are therefore both formal and architectural. Formally, we establish results that constrain the design of verification systems in certain settings. Architecturally, we extract a boundary discipline principle that mandates explicit closure mechanisms external to productive processes when those processes range over sufficiently expressive domains.

The remainder of the paper is organised as follows. Section 2 reviews related work. Section 3 introduces a formal model of verification boundaries and classifies boundary errors. Section 4 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 7 derives architectural consequences for verification-aware systems and formalises a sealing protocol. Section 8 concludes.

## 2 Related Work

The present work engages several distinct literatures. We situate our contributions relative to each in turn.

### 2.1 Verification Architecture and Trusted Computing Bases

The architectural separation between generation and certification has a long history in formal verification. The notion of a trusted computing base, introduced by Anderson [1972] and developed in the "Orange Book" criteria [Department of Defense, 1985], isolates the minimal components on which security guarantees depend. Rushby [1981] articulated design principles for separation kernels that enforce non-interference between components. The central observation in this literature is that trust cannot be distributed arbitrarily; it must be concentrated in components whose correctness can be independently established.

Proof assistants implement this principle through kernel architectures. The LCF approach, originating with Milner [1972] and refined by Gordon et al. [1979], confines proof construction to a small trusted kernel that generates theorems as abstract data types. Tactics and proof search procedures operate outside this kernel and cannot produce theorems except through kernel primitives. Pollack [1998] analysed the rationale for this design, arguing that it permits arbitrarily complex automation while preserving foundational trust. Contemporary systems including Coq [Bertot and Castéran, 2004], Isabelle [Nipkow et al., 2002], and Lean [de Moura et al., 2021] maintain variants of this architecture.

The present paper provides a formal account of why such separation is necessary. The production-closure separation theorem (Theorem 7.4) shows that for sufficiently expressive systems, closure cannot be internalised. This explains the architectural pattern as a response to a structural constraint rather than a mere engineering heuristic.

### 2.2 Hyperintensionality and Fine-Grained Content

The incompatibility result of Section 4 depends on the claim that boundary-constrained consequence is hyperintensional. The study of hyperintensional contexts, in which logically equivalent expressions are not intersubstitutable, has a substantial history in philosophical logic.

Cresswell [1975] introduced the term "hyperintensional" to describe contexts finer-grained than possible-worlds intensions. Berto and Nolan [2017] provide a comprehensive survey of contemporary approaches. The structured propositions tradition, developed by King [2007] and Soames [2010], individuates propositions more finely than sets of possible worlds. Situation semantics [Barwise and Perry, 1983] achieves similar effects by relativising truth to partial information states.

Relevant logic [Anderson and Belnap, 1975] and other substructural logics [Restall, 2000] reject aspects of classical consequence that permit irrelevant premises or conclusions. These systems are motivated by the intuition that valid inference should preserve something beyond mere truth. The present paper shares this intuition but grounds it differently. Our concern is not relevance in general but admissibility relative to boundary constraints.

The closest technical antecedent is the work on awareness logic and explicit knowledge. Fagin and Halpern [1988] introduced awareness structures to model agents who may fail to consider certain propositions. Halpern [2001] applied this framework to game theory. Our Mention Inclusion condition (Condition 4.7) can be understood as a constraint ensuring that conclusions do not presuppose awareness of excluded cases.

### 2.3 Defeasibility and Non-Monotonic Reasoning

The defeasible character of boundary-constrained consequence connects to the literature on non-monotonic reasoning. McCarthy [1980] introduced circumscription as a formalisation of default reasoning. Reiter [1980] developed default logic. Kraus et al. [1990] provided a systematic analysis of non-monotonic consequence relations through postulates.

These systems address the problem that conclusions warranted by incomplete information may be withdrawn when information is added. The present framework addresses a related but distinct phenomenon: conclusions that are classically valid may nonetheless be boundary-inadmissible because their verification requires access to excluded cases.

Makinson [2005] unified various non-monotonic formalisms through the notion of bridges between classical and non-monotonic consequence. The present work can be seen as identifying a particular bridge condition, namely boundary admissibility, that classical consequence fails to respect.

### 2.4 Abstraction, Refinement, and Specification

The boundary errors of Section 3 arise at interfaces between abstraction levels. The general problem of ensuring that abstract specifications correctly characterise concrete implementations is central to formal methods.

Hoare [1972] introduced data abstraction and the use of abstraction functions to relate implementations to specifications. Abadi and Lamport [1991] developed refinement calculi for deriving implementations from specifications through correctness-preserving transformations. Woodcock and Davies [1996] provided a comprehensive treatment of refinement in the Z notation.

A persistent difficulty in this tradition concerns the gap between verified models and deployed systems. Garavel et al. [2019] documented numerous cases where formally verified systems failed in practice due to discrepancies between models and implementations. Klein et al. [2009] addressed this in the seL4 verification by establishing a refinement chain from abstract specification to executable binary.

The present paper provides a conceptual framework for understanding such failures. Interpretation boundary errors (Definition 3.7) arise precisely when the property certified by a verifier does not entail the intended system-level guarantee. This formalises the gap that refinement must bridge.

### 2.5 Probabilistic Reasoning and Uncertainty Quantification

Section 5 addresses the question of whether probabilistic methods can compensate for boundary failures. The relevant background includes Bayesian epistemology [Earman, 1992, Howson and Urbach, 2006] and statistical learning theory [Vapnik, 1998].

Halpern [2003] developed a comprehensive framework for reasoning about uncertainty that combines probability with logic. Pearl [2009] unified probabilistic and causal reasoning. These frameworks are powerful but share a common feature: they operate over a fixed space of possibilities equipped with a fixed event algebra.

The Measure-Partition Mismatch theorem (Theorem 6.1) shows that this feature is limiting. When safety-critical distinctions are not represented in the event algebra, no amount of probabilistic updating can recover them. This observation has antecedents in Hacking [1967] on the problem of old evidence and in discussions of the catch-all hypothesis problem in Bayesian confirmation theory [Shimony, 1970].

Amodei et al. [2016] catalogued concrete risks in machine learning systems and noted that distributional shift undermines probabilistic guarantees. The present paper provides a formal diagnosis: distributional shift is an instance of measure-partition mismatch where the deployment distribution includes events not represented in the training distribution's event algebra.

### 2.6 Reflection and Self-Reference in Formal Systems

The production-closure separation theorem has affinities with classical results on the limits of self-reference. Gödel [1931] showed that sufficiently expressive formal systems cannot prove their own consistency. Tarski [1936] established that truth for a language cannot be defined within that language. Löb [1955] characterised the provability conditions under which a system can derive its own provability statements.

These results concern what systems can prove about themselves. The present result concerns what systems can compute about their own closure status. The connection is not merely analogical. In both cases, the limitation arises from the interaction between expressive power and self-reference.

Harrison [1995] surveyed reflective techniques in theorem proving. Boutin [1997] developed computational reflection in Coq. These techniques permit limited reasoning about proof states within a system. Our result (Section 6.5) explains why such reflection extends productive capacity without eliminating the need for external closure.

# 3 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 verification failures of interest occur 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. We introduce a running example, the SmartAudit pipeline, that we carry through the subsequent formal development.

## 3.1 Verification Pipelines

A verification pipeline consists of a finite sequence of components

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

where each component $C_i$ operates over a domain of artefacts $Adm_i : A_i \to \{\top, \bot\}$. equipped with an admissibility predicate

$$\mathrm{Adm}_i : A_i \rightharpoonup \{\top, \bot\}.$$

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

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

**Definition 3.1** (Pipeline Correctness)**.** A pipeline is *locally correct* if, for each component $C_i$, all artefacts accepted by $\mathrm{Adm}_i$ 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.

## 3.2 The SmartAudit Pipeline

To ground the subsequent formal development, we introduce a concrete verification pipeline that we will use as a running example throughout the paper.

**Example 3.2** (SmartAudit)**.** SmartAudit is a verification pipeline for financial smart contracts. It consists of four stages.

Stage 0 (Specification): A safety property is specified informally as "account balances never become negative." The domain $A_0$ consists of natural language specifications, and $\mathrm{Adm}_0$ checks syntactic well-formedness.

Stage 1 (Formalisation): The specification is formalised as an invariant $\phi$ over a state space $S$ with unsigned 256-bit integer balances. The domain $A_1$ consists of formal specifications, and $\mathrm{Adm}_1$ checks type correctness.

Stage 2 (Verification): An SMT solver attempts to prove that $\phi$ holds for all reachable states. The domain $A_2$ consists of verification queries, and $\mathrm{Adm}_2$ returns $\top$ if the solver reports "valid."

Stage 3 (Deployment): The contract is compiled and deployed to an execution environment using signed 64-bit integers. The domain $A_3$ consists of deployed bytecode, and $\mathrm{Adm}_3$ checks deployment success.

Each stage 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 can fail in the deployed system.

This example illustrates that local correctness does not compose to global correctness. We now characterise the failures that can arise at boundaries.

### 3.3 Boundary Errors

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

**Definition 3.3** (Internal and External Specifications)**.** Each component $C_i$ is associated with:
(i) an internal specification $\Phi_i^{\mathrm{int}}$ defining the property checked by $\mathrm{Adm}_i$, and
(ii) an external specification $\Phi_i^{\mathrm{ext}}$ 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 3.4** (Boundary Error)**.** A *boundary error* occurs at transition $\tau_i$ if there exists an artefact $a \in A_i$ such that:
(i) $\mathrm{Adm}_{i+1}(\tau_i(a)) = \top$,
(ii) $\tau_i(a)$ satisfies $\Phi_{i+1}^{\mathrm{int}}$, and
(iii) $\tau_i$ fails to preserve satisfaction of $\Phi_{i+1}^{\mathrm{ext}}$.

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.

### 3.4 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 3.5** (Embedding Boundary Error)**.** An *embedding boundary* is a transition $\tau_i$ 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.

**Example 3.6** (Embedding Failure in SmartAudit)**.** In the SmartAudit pipeline, the transition $\tau_0$ from natural language specification to formal invariant constitutes an embedding boundary. Suppose the informal specification "balances never become negative" is formalised as $\forall s \in S.\, \mathit{balance}(s) \geq 0$ over unsigned integers. The formalisation silently excludes the possibility of underflow because unsigned integers cannot represent negative values.

This exclusion is an embedding error. The semantic content of "never become negative" includes the possibility of attempted subtractions that would, on signed representations, yield negative results. The formalisation does not capture this content.

### 3.5 Interpretation Boundary Errors

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

**Definition 3.7** (Interpretation Boundary Error)**.** An *interpretation boundary* is a transition at which verified properties must be mapped to external meaning. An *interpretation boundary error* occurs when

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

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

**Example 3.8** (Interpretation Failure in SmartAudit)**.** The transition $\tau_2$ from verified specification to deployed contract constitutes an interpretation boundary. The SMT solver proves that balances remain non-negative in the model using 256-bit unsigned integers. But the deployment environment uses 64-bit signed integers.

The verified property holds for the model. The external specification, that the deployed contract maintains non-negative balances, fails. An integer overflow in the deployment environment can produce a negative balance that the model excludes by construction.

This is an interpretation boundary error. The verifier certifies a property of the model. The certification does not entail the intended guarantee for the deployed system because the interpretation boundary is open.

### 3.6 Verification Layers as Boundary Structure

Many verification pipelines can be abstracted into stages that correspond to the following pattern:

(i) Generative exploration: heuristic production of candidate artefacts.

(ii) Grounding and translation: embedding into formal representations.

(iii) Structural validity: syntactic and type correctness.

(iv) Semantic verification: proof, model checking, or constraint solving.

(v) 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.

### 3.7 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. We will show how the SmartAudit example instantiates this logic.

# 4 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.

## 4.1 Regions and Admissibility

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

**Definition 4.1** (Region)**.** A *region* is a set $R \subseteq W$ representing the possibilities that remain admissible relative to a given verification context.

Verification contexts induce regions by excluding possibilities that violate established constraints. In the SmartAudit example, the formalisation stage induces a region that excludes possibilities involving negative balances by representing balances as unsigned integers.

**Definition 4.2** (Live Region Operator)**.** A *live region operator* is a function

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

satisfying the *shrinkage condition*:

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

Intuitively, $\mathrm{Reg}(\Gamma)$ represents the possibilities consistent with the premises $\Gamma$ under the verification scope currently in force. The shrinkage condition ensures that adding premises does not expand admissibility.

## 4.2 Case Partitions and Boundary Discipline

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

**Definition 4.3** (Case Partition)**.** Given premises $\Gamma$, a family of cases $\mathcal{C} = \{C_1, \ldots, C_n\}$ is a *partition* of $\mathrm{Reg}(\Gamma)$ if:

(i) $\mathrm{Reg}(\Gamma) \subseteq \bigcup_i C_i$ (coverage)
(ii) $C_i \cap C_j = \emptyset$ for $i \neq j$ (disjointness)
(iii) $C_i \subseteq \mathrm{Reg}(\Gamma)$ (in-region)

We refer to the conjunction of these conditions as *boundary discipline*. Coverage ensures that all admissible possibilities are accounted for. Disjointness ensures that cases are not silently conflated.

The role of case partitions is to track which possibilities must be distinguished for correct reasoning, not merely which possibilities are logically compatible with the premises. A consequence relation that respects case partitions will not permit inferences that conflate distinguished cases, even when such conflation would be classically valid.

**Example 4.4** (Case Partition in SmartAudit)**.** In the SmartAudit pipeline, the relevant case partition at the interpretation boundary distinguishes:

$C_1$: States where balance operations remain within the representable range of the deployment environment.

$C_2$: States where balance operations would overflow or underflow in the deployment environment.

The verification model excludes $C_2$ by construction (using unsigned integers that cannot underflow). The deployment environment admits both cases. The boundary error arises because the verification model's region does not partition the deployment environment's possibilities according to this distinction.

### 4.3 Semantic Dependency and Mention

Boundary-constrained reasoning must distinguish between formulae that merely evaluate to the same truth value and formulae whose verification depends on different underlying cases. To capture this distinction, we ground mention sensitivity in semantic dependency.

**Definition 4.5** (Dependency Set)**.** For $\phi \in \mathcal{L}$, define the *dependency set* $\mathrm{Dep}(\phi)$ as the set of worlds $w \in W$ such that there exists $w' \in W$ with $w \models \phi$, $w' \not\models \phi$, and $w$ and $w'$ agree on all atomic propositions except those occurring in $\phi$.

The dependency set contains exactly those worlds across which the truth value of $\phi$ can vary when atomic values are changed. This is a semantic notion, not a syntactic one; it concerns the conditions under which verification of $\phi$ requires distinguishing different possibilities.

**Definition 4.6** (Mentioned Worlds)**.** Let $\mathrm{Mentioned}(\phi)$ be any operator satisfying:

$$\mathrm{Dep}(\phi) \subseteq \mathrm{Mentioned}(\phi) \subseteq W.$$

The Mentioned operator extends the dependency set to include any additional worlds that verification of $\phi$ might require access to. The flexibility in this definition permits different verification contexts to impose different requirements on what must be mentioned.

**Condition 4.7** (Mention Inclusion)**.** For all $\phi \in \mathcal{L}$,

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

Mention Inclusion requires that any case on which the verification of $\phi$ depends must remain admissible when $\phi$ is asserted. This condition expresses the principle that sound verification cannot presuppose access to excluded cases.

The rationale for Mention Inclusion is as follows. Suppose we wish to verify a formula $\phi$ relative to a region $R$. If verifying $\phi$ requires examining whether certain possibilities satisfy $\phi$, those possibilities must be within the region we are reasoning about. A verification procedure that requires access to possibilities outside its scope is not sound relative to that scope, even if it happens to produce the correct answer.

### 4.4 Defeasible Consequence

We now define boundary-constrained defeasible consequence.

**Definition 4.8** (Boundary-Constrained Consequence)**.** Let $\Gamma \subseteq_{\text{fin}} \mathcal{L}$ and $\psi \in \mathcal{L}$. We write

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

if and only if:

(i) $\text{Reg}(\{\psi\}) \subseteq \text{Reg}(\Gamma)$ (boundary admissibility)

(ii) For every case partition $\mathcal{C}$ of $\text{Reg}(\Gamma)$, $\psi$ is true throughout each $C_i \in \mathcal{C}$. (case-respecting necessity)

The first condition enforces boundary admissibility: the conclusion must not require access to possibilities excluded by the premises. The second condition enforces case-respecting necessity: the conclusion must hold throughout the admissible region, uniformly across all distinguished cases.

The second condition is not redundant. A formula might be true throughout $\text{Reg}(\Gamma)$ while failing to respect case structure. For instance, a disjunction might be true throughout a region because each disjunct is true on some cases, without any single disjunct being true throughout. Case-respecting necessity rules out such "gerrymandered" truths when the case partition matters for verification.

**Example 4.9** (Boundary-Constrained Consequence in SmartAudit)**.** Let $\phi$ be the invariant "all balances are non-negative" and let $\psi$ be the proposition "no balance operation underflows." In the verification model with unsigned integers, both $\phi$ and $\psi$ are true throughout $\text{Reg}(\Gamma)$ because underflow is impossible by construction.

However, $\text{Mentioned}(\psi)$ includes the cases where underflow would occur if balances were represented differently. These cases are outside $\text{Reg}(\Gamma)$. Therefore $\Gamma \nvdash_{\text{def}} \psi$; the conclusion is boundary-inadmissible.

This explains the SmartAudit failure. The verification model derives "no underflow" as a consequence. But this conclusion requires access to cases the model excludes. When the deployment environment admits those cases, the derived guarantee fails.

## 4.5 Hyperintensionality

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

**Example 4.10.** Let $\chi = C \to \bot$ for some atomic proposition $C$ that is excluded by boundary constraints, so that $\text{Reg}(\Gamma) \cap \{w : w \models C\} = \emptyset$.

Classically, $\chi$ is true throughout $\text{Reg}(\Gamma)$ because $C$ never holds there. The classical tautology $\top$ is also true throughout $\text{Reg}(\Gamma)$.

However, verification of $\chi$ requires access to the excluded case $C$ in order to discharge the implication. One cannot verify that $C \to \bot$ holds without considering what happens when $C$ holds. Under Mention Inclusion, this means $\text{Mentioned}(\chi) \supseteq \{w : w \models C\}$, which is not contained in $\text{Reg}(\Gamma)$.

Therefore $\Gamma \vdash_{\text{def}} \top$ but $\Gamma \nvdash_{\text{def}} \chi$, despite their classical equivalence over $\text{Reg}(\Gamma)$.

This example is central to the incompatibility result in Section 4.4. It shows that the distinction between boundary-admissible and boundary-inadmissible conclusions cuts across classical equivalence classes.

### 4.6 Relevance Congruence

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 4.11** (Relevance Congruence)**.** A *relevance congruence* $\sim_R$ is an equivalence relation on $\mathcal{L}$ satisfying:

(i) If $\phi \sim_R \psi$, then $\phi \leftrightarrow \psi$ is a tautology.

(ii) If $\phi \sim_R \psi$, then $\mathrm{Reg}(\{\phi\}) = \mathrm{Reg}(\{\psi\})$.

Quotienting by $\sim_R$ permits harmless boundary collapses while preserving formal discipline. Relevance congruence provides a mechanism for engineering judgment to enter the formal framework without abandoning boundary discipline entirely. We return to this relaxation when discussing architectural enforcement mechanisms in Section 7.

## 5 Incompatibility with Extensional Consequence

The purpose of this section is to establish that no sentential consequence relation satisfying standard extensional replacement principles can be boundary-sound. This result explains why verification frameworks typically restrict logical expressivity, track case structure explicitly, or enforce syntactic discipline at boundaries.

### 5.1 Extensional Consequence Relations

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

(i) **Soundness.** If $\Gamma \vdash \phi$, then $\Gamma \models \phi$.

(ii) **Tautological Completeness.** If $\models \phi$, then $\vdash \phi$.

(iii) **Replacement of Equivalents.** If $\models \phi \leftrightarrow \psi$, then for all $\Gamma$, $\Gamma \vdash \phi$ implies $\Gamma \vdash \psi$.

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.

### 5.2 Boundary-Constrained Soundness

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

**Definition 5.1** (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

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

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

### 5.3 The Negative Theorem

**Theorem 5.2** (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.

### 5.4 Proof

We present two proofs. The first uses the vacuity phenomenon and does not depend on details of syntactic representation. The second exhibits a disjunctive case that may be more intuitive.

*First Proof: Vacuity.* Let $C$ be an atomic proposition excluded by boundary constraints, so that

$$\mathrm{Reg}(\Gamma) \cap \{w : w \models C\} = \emptyset.$$

Such exclusions arise routinely in verification through abstraction, modelling assumptions, or environmental constraints.

Let $\phi = C \to \bot$. Classically, $\phi$ is true throughout $\mathrm{Reg}(\Gamma)$, since $C$ never holds there. The implication is vacuously satisfied.

By classical soundness and the assumption that $\Gamma$ is consistent, $\Gamma \models \phi$. By Soundness and Tautological Completeness applied to $\Gamma \cup \{\neg C\}$, we can derive $\phi$ in the classical calculus. More directly, $\phi$ is classically equivalent to $\neg C$, and since $\Gamma \models \neg C$ throughout $\mathrm{Reg}(\Gamma)$, we have $\Gamma \vdash \phi$ in any sound and complete classical system.

However, verification of $\phi$ requires access to the excluded case $C$. To establish that $C \to \bot$ holds, one must verify that no $C$-world satisfies the relevant constraints. This verification procedure requires considering worlds where $C$ holds, even if the conclusion is that such worlds are excluded.

Under Mention Inclusion, this means:

$$\{w : w \models C\} \subseteq \mathrm{Mentioned}(\phi) \subseteq \mathrm{Reg}(\{\phi\}).$$

Since $\{w : w \models C\} \not\subseteq \mathrm{Reg}(\Gamma)$ by assumption, we have $\mathrm{Reg}(\{\phi\}) \not\subseteq \mathrm{Reg}(\Gamma)$.

Therefore $\vdash$ derives $\phi$ while $\phi$ violates boundary admissibility. The consequence relation is not boundary-sound. □

*Second Proof: Disjunctive Expansion.* Let $W = \{w_A, w_B, w_{AB}\}$ where

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

Define a live region operator Reg such that $\mathrm{Reg}(\Gamma) = \{w_A, w_B\}$. Intuitively, the joint case $A \wedge B$ has been excluded by upstream boundary constraints.

Let $\chi = A \vee B$ and $\psi = (A \vee B) \vee (A \wedge B)$. Classically, $\chi$ and $\psi$ are logically equivalent.

By construction, $\chi$ is true throughout $\mathrm{Reg}(\Gamma)$. The dependency set of $\chi$ consists of worlds where the truth of $\chi$ varies with changes to $A$ or $B$. Since $\chi$ is true at $w_A$ (because $A$ holds), at $w_B$ (because $B$ holds), and at $w_{AB}$ (because both hold), and false only at worlds where neither holds, we have $\mathrm{Dep}(\chi) \subseteq \{w_A, w_B\} = \mathrm{Reg}(\Gamma)$ when we restrict to $W$.

However, $\psi$ explicitly involves the joint case $A \wedge B$. The dependency set of $\psi$ includes $w_{AB}$ because the evaluation of the subformula $A \wedge B$ depends on whether both conjuncts hold. Verification of whether $\psi$ holds at various worlds requires considering the joint case, even if $\psi$ happens to be true there.

Under Mention Inclusion, $\mathrm{Mentioned}(\psi) \supseteq \mathrm{Dep}(\psi) \ni w_{AB}$, so:

$$\mathrm{Reg}(\{\psi\}) \supseteq \{w_{AB}\} \not\subseteq \mathrm{Reg}(\Gamma).$$

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

Thus $\vdash$ derives $\psi$ from $\Gamma$ despite $\psi$ violating boundary admissibility. □

### 5.5 Discussion

The two proofs illuminate different aspects of the incompatibility.

The vacuity proof shows that the incompatibility arises from the interaction between extensional validity and boundary admissibility itself. Vacuously true implications are classically valid but require access to excluded cases for their verification. This mirrors practical failures where unreachable branches nevertheless generate verification obligations outside the intended scope.

The disjunctive proof shows that logically redundant syntactic structure can force admissibility commitments. This is perhaps less fundamental but more intuitive: adding a conjunct to a disjunction cannot change its truth value but can change what must be considered in its verification.

The common thread is that extensional replacement permits substitution of expressions that differ in their verification requirements. A boundary-sound consequence relation must track these requirements, which extensional equivalence does not capture.

### 5.6 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 employ one or more of the following strategies: restricting logical expressivity to fragments where the incompatibility does not arise; tracking case structure explicitly through dependent types, refinement types, or similar mechanisms; enforcing syntactic discipline at boundaries to prevent inadmissible reformulations.

The SmartAudit example illustrates the practical consequence. The verification model derives "no underflow" as a consequence of the unsigned integer representation. This conclusion is classically valid within the model. But its verification requires considering what happens when underflow would occur, which the model excludes. When the deployment environment admits those cases, the derived guarantee fails.

## 6 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 when the relevant distinctions are not represented in the probability space. The limitation is structural and persists independently of data volume, model capacity, or update rule.

### 6.1 Probabilistic Inference over Fixed Event Algebras

Let $(\Omega, \mathcal{F}, P)$ be a probability space. In probabilistic verification settings, $\Omega$ represents a space of behaviours or hypotheses, $\mathcal{F}$ a $\sigma$-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 $\Omega$.

Probabilistic inference therefore performs weighting over a fixed ontology. The event algebra $\mathcal{F}$ encodes the distinctions available for probabilistic reasoning. Distinctions not in $\mathcal{F}$ are invisible to probabilistic methods operating on that space.

## 6.2 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 the conditions of Definition 4.3. 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.

## 6.3 Measure-Partition Mismatch

**Theorem 6.1** (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 $E \neq C_k$, $\Delta$ is non-empty. The probability space cannot distinguish elements of $C_k \setminus E$ from elements of $E \setminus C_k$, since this distinction requires an event not in $\mathcal{F}$.

Bayesian updating may drive $P(E)$ arbitrarily close to 1 or 0. However, convergence of $P(E)$ does not eliminate $\Delta$. The distinction required to certify $C_k$ is not expressible in $\mathcal{F}$ and therefore cannot be recovered by probabilistic means.

More precisely, let $E^*$ be the $\mathcal{F}$-measurable event that best approximates $C_k$ in the sense of minimising $P(E^* \triangle C_k)$ for some reference measure. Even if $P(E^*) \to 1$ under updating, the probability mass assigned to $E^* \setminus C_k$ and $C_k \setminus E^*$ remains uncontrolled by the updating procedure, since these sets are not distinguished by $\mathcal{F}$. □

## 6.4 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*. A probabilistically blind certifier assigns high confidence to a verdict while being structurally incapable of detecting the feature that would falsify it.

**Example 6.2** (Probabilistic Blindness in SmartAudit)**.** Consider a probabilistic verification approach to SmartAudit. A machine learning system is trained to predict whether smart contracts satisfy safety properties, using features derived from contract source code and verification model outputs.

The training distribution is generated from the verification model with unsigned integers. The event algebra $\mathcal{F}$ is induced by features computable from this model. The distinction between "safe under 256-bit unsigned semantics" and "safe under 64-bit signed semantics" is not represented in $\mathcal{F}$, since the training data does not include contracts executed under signed semantics.

The probabilistic certifier may achieve high accuracy on contracts similar to the training distribution. It may assign $P(\text{safe}) = 0.99$ to a contract that will fail under deployment semantics. This is not a failure of model capacity or training data quantity. It is a structural impossibility: the distinction required for correct certification is not in the event algebra.

### 6.5 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.

This result has implications for the use of machine learning in verification pipelines. Learned components can accelerate search and improve coverage, but they cannot substitute for boundary-aware certification. The event algebra of a learned model is determined by its training distribution and feature representation. Boundary-critical distinctions not captured in this representation remain invisible to the model regardless of its capacity or the volume of training data.

## 7 Production-Closure Separation

This section establishes that closure cannot be internalised as a finite composition of productive operations in systems of sufficient expressive power. The result explains the architectural necessity of separating generative mechanisms from certifying kernels. It applies to systems where the relevant closure property is undecidable; for systems operating in decidable fragments, internalisation may be possible.

### 7.1 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 7.1** (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. Examples include tactic application in proof assistants, symbolic execution steps, and constraint propagation in solvers.

**Definition 7.2** (Closure Operation)**.** A *closure operation* is a predicate

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

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

Closure determines when productive exploration may soundly stop. In a proof assistant, closure corresponds to having a complete proof. In a model checker, closure corresponds to exhaustive state space coverage.

## 7.2 Internalisation Hypothesis

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

**Definition 7.3** (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 $s \in S$,

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

Under this hypothesis, closure status is decidable by applying $F$ and checking whether the result is closed.

## 7.3 Separation Theorem

**Theorem 7.4** (Production-Closure Separation)**.** *There exist state spaces $S$ and closure predicates $\mathcal{C}$ such that no finite composition of productive operations internalises $\mathcal{C}$.*

*Proof.* Let $S$ be the space of proof states for a sufficiently expressive deductive system, such as first-order arithmetic or a dependent type theory capable of encoding arithmetic. Let $\mathcal{C}(s) = \top$ 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 $\mathcal{C}(F(s)) = \top$ exactly when $s$ is completable to a proof.

Since $F$ is computable from productive operations, and each productive operation is a computable transformation, $F$ itself is computable. Checking whether $\mathcal{C}(F(s)) = \top$ is therefore decidable.

This yields a decision procedure for proof completeness: given any partial proof state $s$, compute $F(s)$ and check closure. If $\mathcal{C}(F(s)) = \top$, the proposition is provable from $s$; otherwise, it is not.

For sufficiently expressive systems, proof completeness is undecidable. By Gödel's incompleteness theorems and the undecidability of the halting problem, there is no algorithm that decides whether an arbitrary partial proof state can be completed.

Hence no such $F$ can exist. Closure cannot be internalised as a finite composition of productive operations in such systems. □

## 7.4 Scope and Limitations

The theorem establishes a limitation for systems of sufficient expressive power. It does not apply uniformly to all verification settings.

For systems operating in decidable fragments, closure may be internalisable. SMT solvers for quantifier-free linear arithmetic, model checkers for finite-state systems, and type checkers for decidable type theories can, in principle, internalise their own closure conditions. The productive exploration terminates, and closure coincides with termination in an accepting state.

The theorem applies when the state space ranges over an undecidable domain. This includes proof assistants for expressive logics, symbolic execution of Turing-complete programs, and any verification setting where the completeness question is undecidable.

The practical significance is as follows: systems that aspire to verify properties of arbitrary programs, proofs, or specifications will encounter the separation. They cannot rely on productive exploration alone to determine closure.

### 7.5 Reflection and Partial Internalisation

Reflective techniques allow limited reasoning about states within the system. Coq's `Ltac` language, for instance, permits tactics that inspect proof state and make decisions accordingly. Computational reflection allows verified programs to operate on representations of terms.

However, reflective procedures are themselves productive operations. Their outputs must still be validated by an external closure predicate. A tactic that claims to have found a proof must submit that proof to the kernel, which checks closure.

Reflection therefore enlarges the productive layer but does not eliminate the need for closure. The separation persists. More powerful reflection permits more sophisticated search and automation, but the final closure check remains external to the productive apparatus.

**Example 7.5** (Reflection in SmartAudit)**.** In a hypothetical reflective SmartAudit system, tactics could inspect the verification model and propose proof strategies. A reflective procedure might recognise common contract patterns and apply specialised verification techniques.

Regardless of the sophistication of these tactics, the final verdict must pass through a trusted kernel that checks whether the proof obligation is actually discharged. The tactics are productive; they cannot internally certify closure.

### 7.6 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 over an undecidable domain must rely on a closure operation that is not reducible to productive exploration itself. The productive layer can grow arbitrarily sophisticated, but it cannot subsume the certifying layer without collapsing into an undecidable procedure.

## 8 Architectural Consequences and Sealing Protocols

The preceding sections establish formal limits on internal certification. This section derives architectural consequences and formalises an explicit sealing protocol that enforces production-closure separation.

### 8.1 The Boundary Discipline Principle

We formulate the central architectural prescription as follows.

> **Boundary Discipline Principle.** Verification architectures must explicitly identify boundaries, enforce admissibility constraints at each boundary, and maintain separation between productive components and closure mechanisms.

This principle is a design recommendation grounded in the formal results of the preceding sections. It does not claim that all systems will naturally evolve toward boundary discipline, nor that boundary discipline is a correctness condition for all verification tasks. Rather, it identifies necessary conditions for reliable verification in settings where the formal limits apply.

The principle has three components: boundary identification, admissibility enforcement, and production-closure separation. We address each in turn.

## 8.2 Boundary Identification

Boundary identification requires making explicit the transitions at which artefacts move between admissibility regimes. In the SmartAudit example, the relevant boundaries are:

(i) The transition from natural language to formal specification (embedding boundary).

(ii) The transition from formal specification to verification query (embedding boundary).

(iii) The transition from verified model to deployed system (interpretation boundary).

Failure to identify boundaries leaves them implicit, and implicit boundaries cannot be systematically checked. The first step in boundary discipline is an audit of the verification pipeline to enumerate all transitions and classify them as embedding or interpretation boundaries.

## 8.3 Admissibility Enforcement

Admissibility enforcement requires explicit mechanisms that check whether artefacts crossing a boundary satisfy the target admissibility conditions.

For embedding boundaries, this means verifying semantic preservation. The formalisation must capture the relevant content of the specification. The verification query must accurately represent the formalisation.

For interpretation boundaries, this means verifying that the certified property entails the intended guarantee in the deployment context. This typically requires explicit assumptions about the deployment environment and checks that these assumptions hold.

**Example 8.1** (Admissibility Enforcement in SmartAudit)**.** To enforce admissibility at the SmartAudit interpretation boundary, the pipeline must:

(i) Explicitly document the assumption that the deployment environment uses 256-bit unsigned integers.

(ii) Check this assumption at deployment time.

(iii) Reject deployment to environments that violate the assumption.

Alternatively, the verification model must be extended to cover the actual deployment semantics. Either approach makes the boundary explicit and enforceable.

## 8.4 Production-Closure Separation

Production-closure separation requires architectural isolation between productive components and closure mechanisms. This is the LCF principle generalised beyond proof assistants.

The productive layer may include heuristic search, machine learning, probabilistic ranking, and arbitrary automation. These components propose candidates and guide exploration.

The closure layer checks finality. It must be simple enough to be trusted independently of the productive layer. In a proof assistant, this is the kernel. In a verification pipeline, this is the final certification stage.

The separation must be enforced architecturally, not merely documented. Productive components must not be able to bypass closure checks or inject uncertified artefacts into the certified output.

## 8.5 Sealing Protocol

We formalise production-closure separation through an explicit sealing protocol.

**Definition 8.2** (Sealing Protocol)**.** A *sealing protocol* for a verification pipeline consists of:

(i) A *sealed type* $T_{\text{sealed}}$ whose inhabitants represent certified artefacts.

(ii) A *seal operation* $\mathsf{seal} : T_{\text{raw}} \to \text{Option}(T_{\text{sealed}})$ that checks closure conditions and, if satisfied, produces a sealed artefact.

(iii) An *architectural guarantee* that $T_{\text{sealed}}$ inhabitants can only be constructed through $\mathsf{seal}$.

The architectural guarantee is enforced through module boundaries, type abstraction, or hardware isolation, depending on the trust model.

The sealing protocol makes production-closure separation explicit and checkable. Productive operations work with raw artefacts. Only the seal operation produces certified artefacts. The type system or module system enforces the separation.

**Example 8.3** (Sealing in Proof Assistants)**.** In Coq, the sealed type is `thm`, representing proved theorems. The seal operation is kernel type checking. The architectural guarantee is that `thm` is an abstract type whose constructors are only accessible to the kernel.

Tactics produce proof terms of type `term`. These are raw artefacts. Only `Qed` invokes the kernel to seal the term into a theorem. If the term does not type-check, sealing fails.

**Example 8.4** (Sealing in SmartAudit)**.** A sealed SmartAudit pipeline would define:

$T_{\text{sealed}}$: Certified contract deployments, represented as pairs of bytecode and certificates.

$\mathsf{seal}$: A function that checks (i) the verification model matches the deployment environment, and (ii) the SMT solver has certified the invariant.

Architectural guarantee: Deployment infrastructure only accepts $T_{\text{sealed}}$ values. Raw contracts cannot be deployed.

This architecture ensures that boundary conditions are checked before deployment and makes the certification status explicit in the type of deployed artefacts.

## 8.6 Relevance Congruence in Architectural Context

The relevance congruence of Definition 4.11 provides a mechanism for engineering judgment within the formal framework. In architectural terms, relevance congruence corresponds to explicit specification of which distinctions matter at each boundary.

A verification pipeline may document that certain distinctions are irrelevant for a given deployment context. For instance, the distinction between different but semantically equivalent bytecode representations may be declared irrelevant if the execution environment treats them identically.

Such declarations must be explicit and auditable. The relevance congruence becomes part of the boundary specification. Changes to deployment context that invalidate the congruence require re-certification.

## 8.7 Machine Learning in Verification Pipelines

The Measure-Partition Mismatch theorem constrains the role of machine learning in verification. Learned components can accelerate productive exploration but cannot substitute for boundary-aware closure.

In architectural terms, machine learning components belong to the productive layer. They may rank candidates, suggest tactics, predict likely invariants, or guide search. Their outputs are raw artefacts subject to sealing.

The closure layer must not depend on learned components for its correctness guarantees. This means that the seal operation must not invoke machine learning, and the architectural guarantee must not rely on properties of learned models.

This constraint is compatible with sophisticated uses of machine learning in verification. The productive layer can be arbitrarily intelligent. The closure layer must be mechanically checkable.

# 9 Conclusion

This paper has established that the separation between generation and certification in verification systems reflects structural limits on internal certification rather than contingent engineering choices.

The boundary error analysis shows that correctness failures concentrate at boundaries between pipeline stages. The incompatibility theorem shows that extensional consequence relations cannot respect boundary admissibility. The Measure-Partition Mismatch theorem shows that probabilistic methods cannot repair ontological misalignment. The production-closure separation theorem shows that closure cannot be internalised as finite productive composition in sufficiently expressive systems.

These results have immediate architectural consequences. Verification pipelines must identify boundaries explicitly, enforce admissibility at each boundary, and maintain production-closure separation through sealing protocols or equivalent mechanisms.

The limits we identify are not universal. They apply to systems of sufficient expressive power and to boundaries where the relevant distinctions are not already represented. For decidable fragments and well-aligned boundaries, the limits do not bite. The contribution is to characterise precisely when and why the limits apply.

Several questions remain open. The relationship between boundary discipline and compositional verification deserves further investigation. The interaction between relevance congruence and formal refinement is not fully characterised. The application of these results to the verification of machine learning systems is a promising direction for future work.

The underlying philosophical point is simple. Production and certification are different activities with different structural properties. Production extends possibilities; certification closes them. No amount of productive ingenuity can substitute for the independent check that establishes closure. This is not a limitation to be overcome but a structural feature to be respected.

# References


Abadi, M. and Lamport, L. (1991). The existence of refinement mappings. *Theoretical Computer Science*, 82(2):253–284.

Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565*.

Anderson, J. P. (1972). Computer security technology planning study. Technical Report ESD-TR-73-51, Electronic Systems Division, Air Force Systems Command.

Anderson, A. R. and Belnap, N. D. (1975). *Entailment: The Logic of Relevance and Necessity*, volume 1. Princeton University Press.

Barwise, J. and Perry, J. (1983). *Situations and Attitudes*. MIT Press.

Bertot, Y. and Castéran, P. (2004). *Interactive Theorem Proving and Program Development: Coq'Art: The Calculus of Inductive Constructions*. Springer.

Berto, F. and Nolan, D. (2017). Hyperintensionality. In Zalta, E. N., editor, *Stanford Encyclopedia of Philosophy*. Spring 2017 edition.

Boutin, S. (1997). Using reflection to build efficient and certified decision procedures. In *Theoretical Aspects of Computer Software*, pages 515–529. Springer.

Cresswell, M. J. (1975). Hyperintensional logic. *Studia Logica*, 34(1):25–38.

Department of Defense (1985). *Trusted Computer System Evaluation Criteria*. DoD 5200.28-STD.

Earman, J. (1992). *Bayes or Bust? A Critical Examination of Bayesian Confirmation Theory*. MIT Press.

Fagin, R. and Halpern, J. Y. (1988). Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34(1):39–76.

Garavel, H., ter Beek, M. H., and van de Pol, J. (2019). The 2020 expert survey on formal methods. In *Formal Methods for Industrial Critical Systems*, pages 3–69. Springer.

Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38:173–198.

Gordon, M., Milner, R., and Wadsworth, C. (1979). *Edinburgh LCF: A Mechanised Logic of Computation*. Springer.

Hacking, I. (1967). Slightly more realistic personal probability. *Philosophy of Science*, 34(4):311–325.

Halpern, J. Y. (2003). *Reasoning About Uncertainty*. MIT Press.

Halpern, J. Y. (2001). Alternative semantics for unawareness. *Games and Economic Behavior*, 37(2):321–339.

Harrison, J. (1995). Metatheory and reflection in theorem proving: A survey and critique. Technical Report CRC-053, SRI Cambridge.

Hoare, C. A. R. (1972). Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281.

Howson, C. and Urbach, P. (2006). *Scientific Reasoning: The Bayesian Approach*. Open Court, 3rd edition.

King, J. C. (2007). *The Nature and Structure of Content*. Oxford University Press.

Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., and Winwood, S. (2009). seL4: Formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles*, pages 207–220.

Kraus, S., Lehmann, D., and Magidor, M. (1990). Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1–2):167–207.

Löb, M. H. (1955). Solution of a problem of Leon Henkin. *Journal of Symbolic Logic*, 20(2):115–118.

Makinson, D. (2005). *Bridges from Classical to Nonmonotonic Logic*. College Publications.

McCarthy, J. (1980). Circumscription—a form of non-monotonic reasoning. *Artificial Intelligence*, 13(1–2):27–39.

Milner, R. (1972). Logic for computable functions: Description of a machine implementation. Technical Report STAN-CS-72-288, Stanford University.

de Moura, L., Kong, S., Avigad, J., van Doorn, F., and von Raumer, J. (2021). The Lean 4 theorem prover and programming language. In *Automated Deduction—CADE 28*, pages 625–635. Springer.

Nipkow, T., Paulson, L. C., and Wenzel, M. (2002). *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Springer.

Pearl, J. (2009). *Causality: Models, Reasoning, and Inference*. Cambridge University Press, 2nd edition.

Pollack, R. (1998). How to believe a machine-checked proof. In *Twenty Five Years of Constructive Type Theory*, pages 205–220. Oxford University Press.

Reiter, R. (1980). A logic for default reasoning. *Artificial Intelligence*, 13(1–2):81–132.

Restall, G. (2000). *An Introduction to Substructural Logics*. Routledge.

Rushby, J. (1981). Design and verification of secure systems. In *Proceedings of the 8th ACM Symposium on Operating Systems Principles*, pages 12–21.

Shimony, A. (1970). Scientific inference. In Colodny, R. G., editor, *The Nature and Function of Scientific Theories*, pages 79–172. University of Pittsburgh Press.

Soames, S. (2010). *What Is Meaning?* Princeton University Press.

Tarski, A. (1936). Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1:261–405.

Vapnik, V. N. (1998). *Statistical Learning Theory*. Wiley.

Woodcock, J. and Davies, J. (1996). *Using Z: Specification, Refinement, and Proof*. Prentice Hall.