What Does "Formally Verified" Actually Guarantee?

A Proof of Structural Limits in Software Verification

Stanislav Komarovsky
Independent Researcher

April 2026

Abstract

We prove that for any formal verification of any real system, the correspondence between the formal proposition and the system it describes cannot be established within any finite tower of formal languages. The proof follows from Tarski's undefinability theorem applied iteratively: verifying that a proposition correctly describes a system requires expressing a correspondence claim that, by Tarski's theorem, cannot be formulated within the proposition's own language. Expressing the correspondence in a richer metalanguage generates a new correspondence claim that cannot be formulated in the metalanguage, producing an infinite regress that no finite extension of the formal framework can resolve. The result is structural, not contingent on current tooling or the complexity of the target system. We discuss five caveats—concerning human knowledge, the value of formal verification, partial gap closure, varying assumption strength, and the functionalist objection—and draw implications for the verification of AI-generated software artifacts.

Keywords: formal verification, Tarski's undefinability theorem, verification regress, software correctness, correspondence problem, AI-generated artifacts, intent evaporation, aboutness gap

---

1 Introduction

Formal verification has emerged as one of the most rigorous approaches to establishing the correctness of software systems. Modern theorem provers—Lean 4, Coq, Dafny, F*—can mechanically check whether a proof establishes a proposition, providing a level of assurance that testing alone cannot deliver. The phrase "formally verified" carries significant weight in both academic and industrial contexts, and its use is expanding as AI systems increasingly generate code, specifications, and even proofs.

Yet the phrase is routinely used as if it establishes more than it does. When a practitioner says a system is "formally verified," the implication is that the system has been proved correct—that it does what it is supposed to do. This paper asks a precise question: can formal verification, in principle, establish that a system does what it is supposed to do? Or is there a structural ceiling beyond which the apparatus of formal proof cannot reach?

We prove that the ceiling exists and is structural. Specifically, we prove that for any formal verification of any real system, there exists at least one correspondence claim—the claim that the formal proposition correctly describes the system—that is necessarily relied upon but cannot itself be formally verified. Moreover, any attempt to verify this claim by extending the formal framework generates a new correspondence claim of the same kind, producing an infinite regress that no finite tower of formal languages can resolve.

The proof follows from Tarski's undefinability theorem (1936), which establishes that the truth predicate for a formal language cannot be defined within that language. We apply this result iteratively to the tower of metalanguages required by successive attempts to formalize the correspondence between a proposition and the system it describes.

This result is not an argument against formal verification. Formal verification remains a valuable and powerful instrument for establishing internal consistency of propositions and proofs. The result is an argument for precision about what the word "verified" means: it means a proposition was checked, not that the proposition describes the system. The gap between the proposition and the system is where human judgment lives, and no formal method can substitute for it.

The result has particular relevance for the verification of AI-generated artifacts. When AI systems generate specifications, code, or proofs, the correspondence between these formal artifacts and the intended system behavior depends on human judgment at every level. Our result establishes that this dependence is not a current limitation of AI capabilities but a structural feature of formal verification itself.

2 Preliminaries

We establish the definitions and prior results on which the main theorem depends.

2.1 Formal Language and Verification

Definition 1 (Formal Language)**.** A *formal language* **L** is a triple (A, F, R) where A is an alphabet of symbols, F is a set of well-formed formulas (propo-

sitions) over A, and R is a set of inference rules that determine which formulas are derivable from which. We require **L** to be sufficiently expressive to encode Peano arithmetic—a standard assumption that holds for all languages used in modern theorem provers (Lean 4, Coq, Dafny, F*, Isabelle/HOL).

Definition 2 (Verification Procedure)**.** A *verification procedure* **V** for a formal language **L** is an algorithm that, given a proposition P in F and a proof object π purporting to establish P, determines whether π is a valid proof of P according to R. In modern type-theoretic provers, this is type-checking: **V** confirms that the proof term π inhabits the type P.

Definition 3 (External System)**.** An *external system* **S** is any entity that exists outside the formal language **L**. **S** may be a software system, a hardware device, a physical process, or any other object about which formal claims are made. We do not require **S** to be formalizable or even fully describable. We require only that **S** exists and is the intended subject of the verification effort.

Definition 4 (Correspondence Claim)**.** A *correspondence claim* $C(P, \mathbf{S})$ is the assertion that a proposition P in **L** correctly describes some aspect of an external system **S**. "Correctly describes" means that the formal property asserted by P holds of **S** when interpreted in the intended way—that is, there exists an interpretation function I mapping the formal symbols of P to entities and relations in **S** such that the truth of P under I coincides with the actual state of affairs in **S**.

2.2 Tarski's Undefinability Theorem

We state Tarski's result in the form required for our proof.

Theorem 1 (Tarski, 1936)**.** *Let* **L** *be a formal language sufficiently expressive to encode Peano arithmetic and its own syntax (via Gödel numbering). There is no formula* $\mathrm{True}(x)$ *in* **L** *such that for every sentence* φ *in* **L**, $\mathrm{True}(\ulcorner\varphi\urcorner)$ *holds in* **L** *if and only if* φ *holds in* **L**, *where* $\ulcorner\varphi\urcorner$ *is the Gödel number of* φ.

Informally: **L** cannot define a truth predicate for its own sentences. To define truth for **L**-sentences requires a metalanguage $\mathbf{L}'$ strictly richer than **L**.

The relevance to verification is this: a correspondence claim $C(P, \mathbf{S})$—"proposition P correctly describes system **S**"—is a truth claim about the relationship between **L**-sentences and entities outside **L**. It requires evaluating whether the formal sentence P, when interpreted over **S**, is true. This is precisely the kind of truth claim that Tarski's theorem shows cannot be expressed within **L** itself.

3 Main Result

We now state and prove the main theorem.

Theorem 2 (Verification Regress)**.** *Let* **L** *be a formal language sufficiently expressive to encode Peano arithmetic. Let* **S** *be any entity external to* **L***. Let* **V** *be a verification procedure in* **L***. Then:*

(i) **V** *can establish that a proof* π *in* **L** *establishes a proposition* P *in* **L***. (Internal verification—decidable within* **L***.)*

(ii) The correspondence claim $C(P, \mathbf{S})$*—"*P *correctly describes* **S***"—is not expressible as a proposition in* **L***.*

(iii) For any extension $\mathbf{L}'$ *of* **L** *in which* $C(P, \mathbf{S})$ *is expressible as a proposition* P' *and provable, a new correspondence claim* $C'(P', \mathbf{S})$ *arises—"*P' *correctly describes the relationship between* P *and* **S***"—which is not expressible in* $\mathbf{L}'$*.*

(iv) Therefore, no finite tower of formal languages $\mathbf{L} \subset \mathbf{L}' \subset \mathbf{L}'' \subset \cdots$ *can establish, by formal means alone, that a proposition in* **L** *correctly describes* **S***.*

Proof. **Part (i).** This follows directly from the definition of **V**. The verification procedure checks whether π is a valid derivation of P according to the inference rules R of **L**. In type-theoretic provers, this amounts to type-checking: confirming that the term π has type P. This is a decidable procedure internal to **L** and is the well-established foundation of all modern proof assistants.

Part (ii). The correspondence claim $C(P, \mathbf{S})$ asserts that P, when interpreted over **S** via an interpretation function I, is true of **S**. This requires **L** to express the relationship between its own sentences and entities external to it—that is, to define a partial truth predicate for **L**-sentences relative to external interpretations.

By Tarski's undefinability theorem, **L** cannot define a truth predicate for its own sentences. *A fortiori*, **L** cannot define a truth predicate for its own sentences relativized to external interpretations, since such a predicate would, for the special case where **S** is the standard model of arithmetic, yield the unrestricted truth predicate for **L**. Therefore $C(P, \mathbf{S})$ is not expressible in **L**.

Part (iii). Suppose we extend **L** to a richer metalanguage $\mathbf{L}'$ that can express claims about the relationship between **L**-sentences and **S**. In $\mathbf{L}'$, the correspondence claim $C(P, \mathbf{S})$ can be formulated as a proposition P' and, supposing the formalization is adequate, proved.

But $\mathbf{L}'$ is itself a formal language sufficiently expressive to encode Peano arithmetic (since $\mathbf{L}'$ extends $\mathbf{L}$, which already encodes it). The proposition P' is an $\mathbf{L}'$-sentence. For the verification of P' to be *about* the actual relationship between P and $\mathbf{S}$—rather than merely about $\mathbf{L}'$-internal objects—a new correspondence claim is needed:

$$C'(P', \mathbf{S}) : \text{“}P' \text{ correctly describes the relationship between } P \text{ and } \mathbf{S}\text{”}$$

By the same argument applied to $\mathbf{L}'$ in place of $\mathbf{L}$, $C'(P', \mathbf{S})$ is not expressible in $\mathbf{L}'$. It requires a further extension $\mathbf{L}''$, and the argument iterates.

Part (iv). Parts (ii) and (iii) establish that each level in the tower $\mathbf{L} \subset \mathbf{L}' \subset \mathbf{L}'' \subset \cdots$ produces a correspondence claim that is not expressible at that level and requires the next level to formalize. Since this holds at every finite level, no finite tower resolves the regress. The gap between the lowest-level proposition P and the external system $\mathbf{S}$ is mediated by an infinite chain of correspondence claims, each requiring a strictly richer language than the one before.

Therefore, for any formal verification effort in any formal language, the claim that the verified proposition correctly describes the external system cannot be established by formal means within any finite extension of the framework. □

Corollary 3. *For any formal verification of any real system, there exists at least one correspondence claim that is relied upon but not formally verified. This claim is a necessary precondition for the verification to be about the system at all, and it cannot be eliminated by any extension of the formal framework.*

Proof. Immediate from Theorem 2. The verification procedure $\mathbf{V}$ establishes that π proves P in $\mathbf{L}$ (Part (i)). For this to constitute verification *of* $\mathbf{S}$, the correspondence claim $C(P, \mathbf{S})$ must hold (Part (ii)). $C(P, \mathbf{S})$ is not provable in $\mathbf{L}$ and generates an infinite regress under any finite extension (Parts (iii)–(iv)). Therefore $C(P, \mathbf{S})$ is necessarily relied upon as an unverified assumption. □

4 Discussion of Caveats

The result above is precise in its scope. Five caveats are necessary to prevent misapplication.

4.1 The Theorem Is About Formal Verification, Not Human Knowledge

Theorem 2 establishes that the correspondence between a formal proposition and a real system cannot be formally verified. It does not establish

that the correspondence cannot be *known*. Humans routinely make correct correspondence judgments—"this model describes that system"—through understanding, testing, domain expertise, code review, integration testing, and accumulated experience. The knowledge that a model describes a system is human knowledge, grounded in empirical engagement with both the model and the system.

The theorem's claim is specifically that this knowledge cannot be formalized and checked by a verification procedure in the way that a proof can be checked against a proposition. The word "verified" in its formal sense does not cover correspondence judgments. In practice, correspondence is established through a combination of informal reasoning, testing, and professional judgment—none of which are captured by the formal apparatus.

4.2 The Theorem Does Not Invalidate Formal Verification

Formal verification establishes that a proof is valid relative to a proposition. This is a real and valuable property. A system whose critical specifications have been formally proved is in a substantially stronger position than one relying on testing alone. The theorem does not argue that formal verification is useless; it argues that the word "verified" should be understood as referring to the proposition, not to the system. Formal verification provides certainty about the formal artifact. The link between the formal artifact and the real system is where uncertainty resides, and acknowledging this is a strength, not a weakness, of the verification practice.

4.3 Partial Closure of Specific Gaps Is Possible

While the full tower of correspondence claims cannot be resolved, individual layers can be tightened or closed. Dafny verifies source code that is then compiled to executable targets; the verified artifact and the running artifact share a common source, reducing the model–implementation gap to the correctness of the Dafny compiler. CompCert provides a verified C compiler, mechanically proved to preserve program semantics from source to assembly, closing one additional layer. seL4 provides a verified microkernel with a machine-checked proof that the C implementation refines the abstract specification.

These projects represent the state of the art in reducing the distance between the formal model and the running system. The theorem does not deny their value. It clarifies that even in these cases, the full tower—from the topmost specification to the physical hardware—is not formally closed. Each project explicitly identifies its trust boundary: seL4 trusts the hardware, CompCert trusts the assembly semantics and the hardware, Dafny trusts its compiler and the target runtime. The correspondence between the low-

est verified layer and the physical world remains a trust assumption, not a formal result.

4.4 The Strength of the Correspondence Assumption Varies

Not all correspondence claims carry the same epistemic weight. When the formal model and the real system are in the same language—as when Dafny verifies Dafny source code that is then compiled—the correspondence assumption is narrow: trust the compiler. When the model is in a different language from the system—as when Lean 4 is used to model Drools rules or Java functions—the correspondence assumption is wide: trust that the human-authored Lean 4 formalization correctly represents the behavior of code in an entirely different language and runtime environment.

The theorem applies equally to both cases, but the practical consequence differs greatly. A narrow correspondence assumption, supported by a verified or extensively tested compiler, is a reasonable engineering practice. A wide correspondence assumption, unsupported by any formal connection between the two languages, is a substantial trust commitment. The theorem makes this distinction visible by identifying exactly where the unverified assumption lies.

4.5 The Functionalist Objection

A functionalist position—following Dennett (1987) and, more recently, Coelho Mollo and Millière (2023, 2025)—holds that intentionality is reducible to computational or causal-functional relations. On this view, the correspondence claim $C(P, \mathbf{S})$ is itself a computational claim and should, in principle, be expressible within a sufficiently rich formal language.

The theorem accommodates this objection. Even granting that $C(P, \mathbf{S})$ is a computational claim expressible in some language $\mathbf{L}'$, Tarski's theorem still applies to $\mathbf{L}'$: the truth predicate for $\mathbf{L}'$ is not definable in $\mathbf{L}'$. The correspondence claim $C'(P', \mathbf{S})$—asserting that the $\mathbf{L}'$-formalization P' of $C(P, \mathbf{S})$ is itself correct—is not expressible in $\mathbf{L}'$. The regress continues regardless of whether intentionality is mystical, physical, or computational. What drives the regress is not the metaphysical nature of correspondence but the self-referential structure of truth for formal languages. The functionalist position does not escape the regress; it reframes what each step in the regress is about without eliminating any step.

5 Implications for AI-Generated Artifacts

The result has particular significance for the growing practice of using AI systems to generate software specifications, code, and formal proofs.

When a human engineer writes a formal specification, the correspondence between the specification and the intended system behavior is grounded in the engineer's understanding of the system, the business requirements, and the domain. This correspondence is not formally verified—Theorem 2 shows it cannot be—but it is supported by a human's genuine engagement with both the formal artifact and the real-world problem. The unverified correspondence assumption is backed by human judgment.

When an AI system generates a specification, no such grounding exists. The AI produces an artifact that is shaped like a specification—syntactically well-formed, formally consistent, and optimized for plausibility based on training data—but the correspondence between this artifact and the intended system behavior is not supported by any being's engagement with the real-world problem. Hattiangadi and Schoubye (2025) characterize such outputs as having "ersatz meaning": they function as if meaningful and can be used to acquire true beliefs, but they do not bear genuine intentionality in the technical sense.

Theorem 2 clarifies why this matters for verification specifically. The correspondence claim $C(P, \mathbf{S})$ was already unverifiable when P was human-authored; the human's judgment was the irreducible trust assumption. When P is AI-generated, the trust assumption is not merely unverified—it is unsupported. No one has the correspondence knowledge that the assumption requires, because no one authored P with an understanding of $\mathbf{S}$.

This has implications across the AI verification landscape. Spec-driven development frameworks (Kiro, Spec Kit, Tessl, OpenSpec) that use AI to generate specifications from natural-language descriptions create artifacts whose correspondence to business intent is mediated by a natural-language-to-formal-language conversion with no formal guarantee of fidelity. LLM evaluation frameworks that use AI-generated rubrics or AI judges introduce correspondence assumptions about the rubric's relationship to the evaluation intent. Governance frameworks that accept AI-generated documentation of "intended purpose" are relying on correspondence claims that no human may have examined.

In each case, the pattern we call *intent evaporation*—the systematic conversion of human purpose into a measurable proxy, with the conversion treated as lossless when it is not—is amplified by AI generation. The proxy was already an imperfect representation of intent; when the proxy itself is generated by a system with no understanding of the intent, the distance between the formal artifact and the human purpose it is supposed to serve becomes unmeasurable within the formal framework.

6 Related Work

The question of whether formal verification can establish the correctness of real software systems has a long and sometimes contentious history.

Fetzer (1988) argued in "Program Verification: The Very Idea" that program verification cannot establish the correctness of programs because programs are causal models of physical processes and the correspondence between the model and the process is empirical, not formal. His argument was controversial and generated vigorous responses from Hoare (1994), who maintained that formal methods provide meaningful assurance despite the model-world gap. Our result is more precise than Fetzer's: it identifies the specific mechanism (the Tarskian regress) that makes the model-world gap formally unclosable, rather than relying on a general argument about the empirical nature of causation.

DeMillo, Lipton, and Perlis (1979) argued that mathematical proof in practice is a social process and that formal verification of programs lacks the social mechanisms (peer review, incremental understanding, community validation) that give mathematical proof its epistemic force. Our result is orthogonal: we do not question the social epistemology of proof but identify a formal limitation that holds regardless of the social context.

The seL4 project (Klein et al., 2009, 2014) represents the most rigorous practical engagement with the model-system gap, providing a chain of machine-checked proofs from an abstract specification through a Haskell prototype to a C implementation, with an explicit trust boundary at the hardware. CompCert (Leroy, 2009) provides a verified compiler with a machine-checked proof of semantic preservation. These projects demonstrate that individual layers of the correspondence can be formally closed, consistent with Caveat 4.3 above.

Tarski's undefinability theorem (1936) and Gödel's incompleteness theorems (1931) are the foundational results on which our argument rests. The specific application of Tarski's theorem to the correspondence between formal models and external systems, via an iterative regress argument, appears to be novel in the context of software verification, though the general pattern of Tarskian hierarchies is well-known in mathematical logic (Kripke, 1975; Halbach, 2014).

Hattiangadi and Schoubye (2025) argue that LLM outputs are "meaningless" in the technical sense that they lack communicative intentions, coining the term "ersatz meaning." Storey et al. (2026) propose "intent debt" as a distinct category of software health debt. Both works approach the territory addressed by our result from different angles—philosophy of language and software engineering, respectively—without formalizing the verification-specific impossibility.

Rice's theorem (1953) establishes that non-trivial semantic properties of programs are undecidable, blocking general verification algorithms. Our result is complementary: Rice shows that verification cannot be automated for arbitrary programs; we show that even when verification succeeds for a specific program (via a specific proof), the result does not extend to the real system without an unverifiable correspondence assumption.

7 Conclusion

We have proved that for any formal verification of any real system, the claim that the formal proposition correctly describes the system cannot be established within any finite tower of formal languages. The result follows from Tarski's undefinability theorem applied iteratively and is structural—it does not depend on the complexity of the target system, the sophistication of the verification tool, or the quality of the formal model.

The practical consequence is not that formal verification is valueless but that the word "verified" should be understood precisely. Formal verification establishes that a proof is valid relative to a proposition. Whether the proposition describes the system is a correspondence judgment that lies outside the formal apparatus. This judgment is typically sound when made by competent engineers with domain expertise and access to the real system. It is not sound when no such engineer exists—as is increasingly the case when AI systems generate specifications, code, and proofs without human engagement with the underlying problem.

The discipline required by this result is not better verification. It is precision about what verification can and cannot do, and the recognition that the correspondence between formal artifacts and the real world is, and will remain, a human responsibility.

A machine-checked formalization of the main theorem in Lean 4 or a similar proof assistant, while beyond the scope of the present paper, would be valuable future work and would itself serve as an illustration of the result: the formalization would verify the theorem's internal consistency while the correspondence between the formalized theorem and the metatheoretic claim it represents would remain a human judgment.

References

Coelho Mollo & Millière (2023/2025)
The Vector Grounding Problem. *PhilArchive*.

DeMillo, Lipton & Perlis (1979)
Social Processes and Proofs of Theorems and Programs. *Communications of the ACM*, 22(5), 271–280.

Dennett (1987)
The Intentional Stance. MIT Press.

Fetzer (1988)
Program Verification: The Very Idea. *Communications of the ACM*, 31(9), 1048–1063.

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

Halbach (2014)
Axiomatic Theories of Truth. Cambridge University Press.

Hattiangadi & Schoubye (2025)
The Outputs of Large Language Models are Meaningless. *arXiv*:2509.22206. Forthcoming, Oxford University Press.

Hoare (1994)
How Did Software Get So Reliable Without Proof? In *FME '96: Industrial Benefit and Advances in Formal Methods*, Springer.

Klein et al. (2009)
seL4: Formal Verification of an OS Kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*.

Kripke (1975)
Outline of a Theory of Truth. *The Journal of Philosophy*, 72(19), 690–716.

Leroy (2009)
Formal Verification of a Realistic Compiler. *Communications of the ACM*, 52(7), 107–115.

Rice (1953)
Classes of Recursively Enumerable Sets and Their Decision Problems. *Transactions of the American Mathematical Society*, 74(2), 358–366.

Storey et al. (2026)
From Technical Debt to Cognitive and Intent Debt: Rethinking Software Health in the Age of AI. *arXiv*:2603.22106.

Tarski (1936)

Der Wahrheitsbegriff in den formalisierten Sprachen. *Studia Philosophica*, 1, 261–405. English translation in Tarski (1956), *Logic, Semantics, Metamathematics*, Oxford University Press.