Article

Not peer-reviewed version

Structural Limitations in the P vs NP Framework: A Formal Reassessment

Michael Cody *

Posted Date: 7 August 2025

doi: 10.20944/preprints202508.0455.v1

Keywords: computational complexity; P vs. NP problem; NP-completeness; polynomial-time reductions; structural complexity; cook–levin theorem; verifier logic; computational hardness; complexity theory foundations; semantic preservation

 doi:10.20944/preprints202508.0455.v1

Article

Structural Limitations in the P vs. NP Framework: A Formal Reassessment

Michael Aaron Cody

Independent Theorist; mac92contact@gmail.com

Abstract

This paper presents a formal structural audit of the classical P vs NP framework, identifying interpretive tensions across verifier logic, reduction structure, and asymptotic analysis. While the formal machinery operates consistently at the syntactic level, this paper show that common extensions, particularly in applications to security, verification, and feasibility, rely on assumptions not entailed by the original definitions. This paper doesn't propose a new model or resolution to the P vs NP problem. Instead, I demonstrate that the framework's compositional use as a proxy for structural hardness introduces semantic boundaries that complicate unified application. This audit highlights the need for more precise separation between decision preservation and constructive complexity.

Keywords: computational complexity; P vs. NP problem; NP-completeness; polynomial-time reductions; structural complexity; cook–levin theorem; verifier logic; computational hardness; complexity theory foundations; semantic preservation

---

1. Introduction

The assumption that $P \neq NP$ is a foundational principle of computational complexity theory [1,12]. It underpins problem classification, intractability arguments, and the boundary between solvable and verifiable problems. Despite decades of research, this assumption remains unproven, maintained by convention rather than formal resolution.

This paper operates entirely within the classical framework: deterministic Turing machines, polynomial-time verifiers, and Karp reductions. No symbolic reinterpretation or alternative computational model is introduced. Instead, it conducts a structural audit, evaluating whether the definitional tools used to support the $P \neq NP$ boundary maintain alignment when applied in composition.

Let $L \in NP$ with verifier V such that:

$$x \in L \iff \exists y \in \{0,1\}^{p(|x|)} \text{ where } V(x,y) = 1$$

By the Cook-Levin theorem [3], there exists a polynomial-time reduction R such that:

$$x \in L \iff R(x) \in \text{SAT}$$

This establishes decision equivalence. However, there is no requirement that any satisfying assignment z for $R(x)$ allows reconstruction of the original witness y via a polynomial function:

$$\neg\exists f \in \text{poly such that } f(z) = y$$

Thus, while membership in L is preserved, semantic continuity between verification and transformation is not.

This does not imply contradiction. It highlights that the components of the NP framework, verifiers, reductions, and completeness logic, are not required to preserve mutual interpretability when

 doi:10.20944/preprints202508.0455.v1

applied sequentially. The result is not a logical failure but a semantic divergence that complicates unified interpretation of the boundary.

Six points of divergence are identified where this gap emerges:

1. **Reduction Structure Loss**: Decision equivalence is preserved, but verifier traceability is not.
2. **Verifier–Transformation Gap**: Satisfiability does not imply witness-level correspondence.
3. **Cook–Levin Flattening**: The encoding removes control and state structure.
4. **Asymptotic Concealment**: Symbolic bounds obscure constant-driven divergence in structure.
5. **Constructibility Dissociation**: Witness existence does not guarantee reconstruction after transformation.
6. **Tool Application Overreach**: Reductions are used to justify structural hardness beyond their formal guarantees.

This paper makes no claim to resolve P vs NP constructively. Rather, it shows that when class definitions, verification semantics, and completeness transformations are composed, gaps emerge between formal guarantees and unified interpretability. This invites reexamination of how completeness is invoked to justify structural hardness in the absence of preserved semantic continuity.

Note: This analysis does not assert a formal contradiction within complexity theory's axioms. Rather, it identifies interpretive tensions that emerge when the framework's tools, verifiers, reductions, asymptotics, are composed beyond their intended boundaries. Our critique is structural and semantic, not logical or axiomatic.

2. Formal Target: The P vs. NP Framework

In classical complexity theory, the class P consists of decision problems solvable in polynomial time by a deterministic Turing machine [12]. A language L belongs to P if there exists a Turing machine M and a polynomial $p(n)$ such that for every input x, $M(x)$ halts in at most $p(|x|)$ steps and correctly decides whether $x \in L$.

The class NP consists of decision problems for which proposed solutions can be verified in polynomial time [1,12]. Formally, $L \in NP$ if there exists a polynomial-time verifier $V(x, y)$ such that:

$$x \in L \iff \exists y \in \{0,1\}^{p(|x|)} \text{ where } V(x, y) = 1$$

The central question of the P vs NP problem is whether $P = NP$; that is, whether every problem for which a solution can be verified in polynomial time can also be solved in polynomial time. The prevailing belief in theoretical computer science is that $P \neq NP$.

This assumption supports the classification of problems such as Boolean satisfiability, integer factorization, discrete logarithms, and circuit minimization as efficiently verifiable but not known to be efficiently solvable [5]. While not all of these problems are formally known to be NP-complete, their exponential worst-case behavior and resistance to known polynomial-time algorithms support their practical intractability.

If $P = NP$, then for all $L \in NP$, there exists a polynomial-time algorithm that, given input x, produces a certificate y such that $V(x, y) = 1$. This would eliminate the classical separation between search, verification, and decision.

This paper accepts the standard definitions of P, NP, polynomial-time verifiers, and reductions. No alternative frameworks are proposed. The analysis examines the formal boundaries of what these tools preserve when applied in sequence. Specifically, it investigates the relationship between verifier acceptance in the original problem and witness recoverability after reduction. This examination is conducted entirely within classical complexity theory to understand the semantic boundaries of compositional tool application.

 doi:10.20944/preprints202508.0455.v1

3. Reduction Fallacy: Failure of Polynomial Equivalence

The class of NP-complete problems is constructed by applying polynomial-time reductions between problems. A problem A is said to be reducible to a problem B if there exists a function f, computable in polynomial time, such that for all x,

$$x \in A \iff f(x) \in B.$$

This reduction, denoted $A \leq_p B$, preserves decision outcomes. If B is solvable in polynomial time, then A becomes solvable in polynomial time through composition with f. This mechanism forms the basis of the Cook-Levin theorem [3,12] and subsequent transitivity results within NP-completeness theory.

This construction assumes that the function f, while altering the instance, does not distort the computational structure of the original problem. That assumption does not hold under empirical examination.

Polynomial-time reductions preserve solvability, but they do not preserve structural properties of the solution space. Specifically, reductions do not maintain:

- The number of candidate solutions
- The structure of witness encodings
- The branching factor of the search space
- The memory layout and state transition behavior
- The computational cost of constructing or extracting a solution

Example: In the standard reduction from 3SAT to CLIQUE [5], each clause becomes a vertex set, and edges are inserted between non-conflicting literals. This produces a graph with $O(k \cdot l)$ vertices and up to $O(k^2 \cdot l^2)$ edges, where k is the number of clauses and l is the number of literals per clause. Although the size increase is polynomial, the number of cliques to evaluate grows significantly compared to the structure of the original formula. The traversal complexity, memory layout, and solution density are all altered.

Theorem 1. *Let S_A be the solution space of problem A, and let S_B be the induced solution space of $f(A)$ for a polynomial-time reduction f. Then:*

$$A \leq_p B \not\Rightarrow |S_A| \approx |S_B| \quad \textit{nor} \quad \textit{Traversal}(S_A) \approx \textit{Traversal}(S_B).$$

That is, polynomial reducibility does not imply preservation of solution space cardinality or access complexity.

This result highlights the distinction between decision equivalence and witness equivalence in polynomial-time reductions.

Therefore, the use of NP-completeness to imply equivalence of difficulty or structure across problems is not valid. NP-complete problems share symbolic decision-preserving reducibility, but not operational fidelity. In practical terms, reductions alter the shape, depth, and cost of search and construction. As such, the NP-complete label does not denote a true equivalence class in the full mathematical sense. It marks decision equivalence, not structural identity.

This limitation affects the design of reduction-based algorithms and the interpretation of hardness results in applied settings.

4. Asymptotic Abstraction and Practical Boundaries

Asymptotic notation, particularly Big-O, is a foundational tool in computational complexity theory [4]. It provides a framework to express the upper bounds on the growth rate of algorithms as input size approaches infinity. This allows for classification of algorithms by their long-run scaling behavior, independent of implementation details or constant factors.

 doi:10.20944/preprints202508.0455.v1

Let $T(n) = c(n) \cdot n^k$ denote the time complexity of an algorithm where $c(n)$ encodes overhead related to recursion depth, memory usage, or constraint processing. Under asymptotic abstraction, $c(n)$ is absorbed into the constant term of the $O(n^k)$ classification. This abstraction is formally correct but may obscure critical structural differences when comparing problems near the complexity class boundaries.

Example: In a 3SAT instance with clause density $d(n)$, a syntactically valid polynomial transformation may increase inter-clause dependencies or logical constraints that alter the effective traversal complexity. These differences are not captured by k but impact the feasibility of deterministic traversal.

In such settings, two decision problems classified as $O(n^k)$ may show divergence in evaluation cost:

$$T_1(n) = c_1(n) \cdot n^k, \quad T_2(n) = c_2(n) \cdot n^k, \quad \text{with } c_1(n) \neq c_2(n).$$

Even when both expressions satisfy the definition of polynomial-time, the internal overhead encoded in $c(n)$ may yield substantially different performance.

This does not imply that Big-O misclassifies problems. The notation correctly expresses asymptotic limits. However, when asymptotic classes are interpreted as proxies for feasibility or tractability, there is potential for mismatch. The formal guarantees hold, but their implications for practical evaluation and implementation require caution.

The distinction is particularly relevant in applied domains such as cryptographic hardness [7], program obfuscation [2], and resource-bounded verification [1]. In these contexts, assumptions about computational hardness are often drawn from NP classification. Such assumptions presume more than decision equivalence; they infer hardness of construction, traversal, or solution extraction, none of which are implied by the Big-O definition.

This section does not critique asymptotic notation itself. Instead, it identifies a boundary. Formal classification provides essential insights into growth rates but abstracts away structural behaviors that influence practical feasibility. Recognition of this interpretive boundary is necessary to align complexity theory with its applications.

5. Verifier Degeneracy in NP Definition

The class NP is formally defined as the set of decision problems for which a proposed solution can be verified in polynomial time by a deterministic Turing machine [1,12]. That is, there exists a verifier function $V(x, y)$ and a polynomial $p(n)$ such that:

$$x \in L \iff \exists y \in \{0,1\}^{p(|x|)} \text{ such that } V(x, y) = 1.$$

This construct separates NP from P by allowing problems to be verifiable in polynomial time, even when no known method exists to compute y directly.

However, this model introduces a critical limitation. The verifier only confirms correctness for an input-certificate pair. It makes no claim regarding how that certificate was obtained. The definition assumes the existence of such a certificate but offers no operational mechanism to recover it. This creates a structural asymmetry between verification and construction.

Formally define:

$$\text{EfficientConstruction}(x) \iff \exists A \in \text{P} \text{ such that } A(x) = y \text{ and } V(x, y) = 1.$$

In many cases, $\text{VerifierExists}(x)$ holds without any known $A \in P$ satisfying $\text{EfficientConstruction}(x)$. The verification model becomes decoupled from any generative algorithm.

Asymmetry Summary:

- *Verification* is explicitly bounded by polynomial-time execution.
- *Existence of solution* is presumed via acceptance without constructive support.
- *Construction* is neither required nor implied by the definition.

 doi:10.20944/preprints202508.0455.v1

This leads to a degenerate boundary condition. The class NP accepts problems based on the verifier's response, but the meaningfulness of that response depends on a certificate whose origin is undefined. The definition succeeds syntactically but fails operationally if no procedure exists to construct witnesses efficiently.

When used to assert problem difficulty, this structure causes ambiguity. Let:

$$\text{ProblemHard}(x) \iff \text{VerifierExists}(x) \wedge \neg\text{EfficientConstruction}(x).$$

This expression captures the informal basis for intractability in applied cryptography and optimization. However, $\neg\text{EfficientConstruction}(x)$ is not demonstrable, it is inferred from absence of discovery. This is a nonconstructive basis for class boundary claims.

If verifier acceptance implies efficient construction, then $P = NP$. If not, then NP permits labeling problems as "intractable" without operational grounding. In both scenarios, the verifier's role lacks definitional independence from assumed hardness.

This does not constitute a contradiction in logic. It constitutes a structural degeneracy. The class definition permits semantic conclusions (e.g., cryptographic hardness, optimization infeasibility) that cannot be directly derived from its own operational model. Without guaranteed constructive paths, the NP class provides bounded checking but does not isolate computational difficulty. The separation from P then exists in formal syntax but not in algorithmic content.

6. Cook–Levin Inversion: Loss of Derivational Fidelity

The Cook–Levin Theorem proves that every language in NP can be reduced to SAT in polynomial time. Given a verifier $V(x, y)$ operating within polynomial time $p(n)$, a Boolean circuit $C_{x,y}$ simulating the computation of V is constructed. This circuit is encoded into a formula ϕ such that:

$$\phi \in \text{SAT} \iff \exists y \text{ such that } V(x, y) = 1.$$

This reduction correctly preserves decision satisfiability. However, it does not preserve structural features of the original verifier computation, particularly recursion fidelity, state traversal identity, and construction topology.

Let $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be the verifier for some language $L \in \text{NP}$. The reduction transforms the configuration history of V into a tableau, a 2D matrix of cell states over time, then encodes this matrix as a SAT formula. Each row of the tableau simulates a computation step, encoded into variables and constraints:

$$\phi(x) = \bigwedge_{t=1}^{p(n)} \bigwedge_{i=1}^{s} C_{i,t}(x),$$

where $C_{i,t}$ encodes the state at cell i, time t, and s is the space bound.

This reduction flattens the verifier's recursive stack trace into a spatial Boolean grid. While this grid preserves satisfiability, it severs the semantic identity between:

- Input x and its verifier path traversal
- Certificate y and its recursive derivation
- State transitions and logical control flow

Let:

$$\psi(x) = \text{SAT reduction of } V(x, y).$$

Then:

$$\exists y : V(x, y) = 1 \Rightarrow \psi(x) \in \text{SAT},$$

but:

$$\text{SAT solution of } \psi(x) \not\Rightarrow \text{Constructive derivation of } y.$$

 doi:10.20944/preprints202508.0455.v1

The structure of the original verifier, symbolic recursion, stack evolution, and depth traversal, is not preserved. The SAT instance encodes a valid solution state, but omits the derivational steps used to reach it.

Recursion fidelity refers to the preservation of:

- The verifier's control flow
- State-dependent transitions
- Input-certificate dependency mappings

This analysis examines what Cook–Levin preserves beyond decision equivalence. The reduction correctly maintains satisfiability; the limitation concerns reconstruction of the original verifier's computational path. This distinction pertains to witness reconstruction difficulty, not decision complexity.

Thus, constructive hardness properties may not transfer directly from L to $\mathrm{SAT}(L)$ due to the loss of derivational semantics:

$$\mathrm{ConstructiveHardness}_L \not\Rightarrow \mathrm{ConstructiveHardness}_{\mathrm{SAT}(L)}.$$

This may have implications for applied systems that rely on SAT-based encodings for synthesis or constraint-solving. Cryptographic primitives, obfuscation models, and system verifiers often assume that problem hardness is preserved through the reduction. However, the SAT formula produced by Cook–Levin does not reconstruct the recursive logic or traversal history of the original verifier. As a result, symbolic completeness is maintained, but constructive alignment is not.

The theorem remains valid, but fidelity loss prevents computational traceability. Completeness without derivational fidelity may satisfy formal equivalence but diverges from operational equivalence in system-critical domains.

7. Compositional Boundaries in Classical Complexity Frameworks

This section synthesizes the structural observations from Sections 3–6 to assess the interpretive and compositional boundaries of the classical P vs NP framework. While each component, reductions, asymptotic analysis, verifier models, and completeness constructions, operates correctly under its own formal guarantees, their interaction reveals semantic drift when applied collectively to real-world tractability, constructibility, and hardness claims.

7.1. Summary of Observed Limitations

B1. **Reductions and Structural Divergence:** Polynomial-time reductions preserve decision boundaries but may alter the geometry, traversal, or certificate density of the original solution space. This limits their applicability for claims about structural or operational equivalence between NP-complete problems.

B2. **Asymptotic Abstraction:** Big-O classification abstracts constant factors and instance-dependent behaviors that may dominate the actual runtime. While the abstraction is valid in theory, it introduces interpretive risk when equated with practical feasibility.

B3. **Verifier–Construction Gap:** The verifier model ensures that valid certificates can be checked efficiently, but does not imply the existence of efficient methods to generate those certificates. This distinction is often underemphasized in applied complexity contexts.

B4. **Witness Recovery in Cook–Levin:** The SAT encoding constructed by Cook–Levin guarantees satisfiability preservation but does not maintain the derivational semantics of the original verifier logic. As a result, solutions to the SAT instance may not provide transparent or reconstructible paths to original problem witnesses.

B5. **Entropy Scaling and State Explosion:** Certain verifier behaviors may exhibit entropy scaling that, while consistent with polynomial verifier time, still imply exponential state space traversal in the underlying certificate domain. This complicates interpretation of "feasibility" based purely on class membership.

 doi:10.20944/preprints202508.0455.v1

7.2. Interpretive Misalignment

These observations do not imply that the P vs NP framework contains contradictions. Rather, they show that:

- The tools operate within well-defined formal boundaries.
- When composed or applied to broader theoretical or practical questions (e.g., constructibility, cryptographic hardness, optimization synthesis), the limits of each tool become more pronounced.
- Some widely held intuitions, such as NP-completeness implying practical hardness, require further scrutiny, especially when witness reconstruction, traversal cost, or solution structure are relevant.

Clarification: This section does not dispute the validity of polynomial-time reductions, verifier models, or completeness theorems. It identifies the boundaries of their guarantees and highlights how composition can introduce gaps between formal classification and applied inference.

7.3. Definition: Semantic Preservation under Reduction

Let $A, B \in \mathsf{NP}$ and $f : A \to B$ a polynomial-time reduction. I define f to be **semantically preserving** if the following hold:

- The number of solution candidates is preserved up to polynomial distortion:

$$|S_A| \in \Theta(|S_B|)$$

- The branching structure and traversal depth of the verifier's computation remains similar:

$$\text{Traversal}_A \sim \text{Traversal}_B$$

- The transformation permits inverse mapping or reconstruction of witnesses:

$$y_B \mapsto y_A \text{ in polynomial time}$$

In most reductions used to prove NP-completeness, only the first-order decision relationship is preserved:

$$x \in A \iff f(x) \in B$$

but the deeper properties above are not. This is not a flaw, it is a boundary.

7.4. Interpretive Drift in Applied Contexts

While the formal tools of complexity theory function as designed, their interpretive application often extends beyond what the definitions guarantee. For instance, SAT-based cryptographic constructions assume that finding satisfying assignments is intractable because SAT is NP-complete [7], yet this presumes that hardness of decision implies hardness of witness construction. Similarly, compiler optimization, secure obfuscation, and AI constraint-solving literature frequently reference NP-completeness as evidence of real-world intractability [1,2], despite using reductions that distort input structure, certificate recoverability, or traversal cost. These claims treat symbolic decision membership as a proxy for structural or operational difficulty, which exceeds the formal guarantees of the P vs NP framework.

7.5. Conclusions

The classical P vs NP framework operates with internal consistency. However, the composition of its components introduces semantic gaps that must be acknowledged when extrapolating from theoretical class membership to practical feasibility or hardness claims. This paper does not propose new definitions or dispute established results. It calls for greater precision in interpreting what the classical tools guarantee, and where their guarantees stop.

 doi:10.20944/preprints202508.0455.v1

Data Availability Statement: All data supporting the findings of this study are theoretical constructs or fully contained within the manuscript. No external datasets were used.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Arora, S., & Barak, B. (2009). *Computational complexity: A modern approach*. Cambridge University Press.
2. Barak, B., et al. (2001). On the (Im)possibility of Obfuscating Programs. *Advances in Cryptology—CRYPTO 2001*, 1–18.
3. Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing* (pp. 151–158). ACM.
4. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
5. Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. W. H. Freeman.
6. Gödel, K. (1931). Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. *Monatshefte für Mathematik und Physik*, 38, 173–198.
7. Goldreich, O. (2008). *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press.
8. Hilbert, D. (1928). Die Grundlagen der Mathematik. In *Verhandlungen des Internationalen Mathematiker-Kongresses 1928* (pp. 171–176). Springer.
9. Kuhn, T. S. (1962). *The Structure of Scientific Revolutions*. University of Chicago Press.
10. Levin, L. A. (1973). Universal search problems. *Problems of Information Transmission*, 9(3), 265–266.
11. Russell, B. (1903). *The Principles of Mathematics*. Cambridge University Press.
12. Sipser, M. (2012). *Introduction to the theory of computation* (3rd ed.). Cengage Learning.