STRUCTURE-AWARE EQUIVALENCE FOR EVALUATING AUTOFORMALIZED STATEMENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating autoformalized mathematical statements requires determining whether a candidate preserves the mathematical claim intended by a trusted reference. Existing methods approach this problem from complementary directions: reference-free learned evaluators estimate semantic agreement but are not constrained to a formal equivalence relation, while reference-based proof methods typically employ whole-statement logical equivalence to provide formal certificates, which is too coarse for translation fidelity and may identify mathematically unrelated statements.

We introduce *Structure-Aware Equivalence* (SAE), a role-preserving formal equivalence relation between syntactic and whole-statement logical equivalence that allows formally justified reformulations while preserving translation-relevant mathematical structure. We operationalize SAE in the *Autoformalization Correctness Evaluator* (ACE), which provides certified equivalence or inequivalence judgments and reports Inconclusive otherwise. On ProofNetVerif, a benchmark introduced with BEq+, ACE retains substantial equivalent-case coverage while remaining highly conservative in certifying disputed or inequivalent pairs. On targeted diagnostics, BEq+ accepts 94–96% of unrelated but whole-statement logically equivalent pairs, while Mathesis LeanScorer accepts 42% of localized meaning-changing perturbations; ACE accepts none of either set as equivalent.

1 INTRODUCTION

Large language models (LLMs) are increasingly capable of translating informal mathematical statements into formal languages such as Lean (Azerbayev et al., 2023; Wu et al., 2022; Weng et al., 2025). Yet measuring whether a generated formal statement is *semantically correct* remains a fundamental bottleneck. A statement may compile successfully while altering its assumptions, mathematical domain, or conclusion. Conversely, a correct formalization need not reproduce a reference statement syntactically: differences in notation, variable names, or algebraic presentation need not change the intended mathematics. Reliable autoformalization therefore requires deciding whether different formal representations preserve the same intended mathematical claim.

There are two complementary settings for this problem. In *reference-free evaluation*, an informal statement and a candidate formalization are available, but no trusted formal counterpart exists. In *reference-based evaluation*, a trusted formalization is available and the candidate is evaluated against it. The former is indispensable for previously unformalized mathematics, while the latter serves a different and practically important role in benchmarking models, curating formal datasets, and providing reliable training or selection signals. Importantly, a reference does not make evaluation trivial: exact matching rejects valid reformulations, while compilation or independent provability does not establish semantic equivalence to the reference.

Reference-free methods such as FormalAlign (Lu et al., 2025) and Mathesis LeanScorer (Yu et al., 2026) use learned semantic alignment or LLM-based judgments to compare natural- and formal-language statements. Such methods are valuable when no trusted formal reference exists, but their decisions are not constrained to realize a formally defined semantic relation. For direct LLM-as-a-judge methods, this weakness can appear visibly as contradictory judgments for the same pair across repeated evaluations or prompt variations (Xie et al., 2024; Wei et al., 2025). More generally, learned evaluators provide no guarantee that semantics-preserving reformulations retain the same judgment,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

or that localized semantics-changing edits alter it. Their outputs are therefore learned predictions of semantic agreement rather than mechanically checkable evidence of that agreement.

A trusted formal reference makes stronger guarantees possible: once both the reference and candidate are Lean statements, their relationship can itself be subjected to formal proof. Reference-based metrics such as BEq and BEq+ attempt to mutually derive a reference T_r and candidate T_c, thereby establishing whole-statement logical equivalence, $T_r \to T_c$ and $T_c \to T_r$ (Liu et al., 2025; Poiroux et al., 2025). When successful, these derivations provide mechanically checked proofs, but the relation itself is too coarse for translation correctness. For an extreme but concrete example, consider

$$(1 : \mathbb{N}) + 1 = 2 \qquad \leftrightarrow \qquad \text{Fermat's Last Theorem.}$$

Both propositions are now formally provable in Lean (Anthropic, 2026), their biconditional is also provable: either implication can simply ignore its assumption and invoke the theorem on the other side. Yet the two clearly do not express the same mathematical statement. The issue is not a failure of theorem proving: the prover correctly certifies the formal relation. Rather, logical equivalence is too coarse to serve directly as translation equivalence.

This distinction also changes how proof-search strength should be interpreted. Incomplete search may hide the mismatch by failing to discover logically valid but translation-inappropriate equivalences, whereas stronger search more faithfully realizes the chosen relation. It cannot, however, repair a misaligned correctness criterion, and failure to establish equivalence is not a proof of inequivalence. BEq+, for example, certifies equivalence when both implication directions are proved, but a failed proof search does not provide a Lean-checked proof of $\neg(T_r \leftrightarrow T_c)$. The choice of *what relation defines correctness* must therefore be separated from the ability of a prover to *certify that relation*.

Nor does the reference itself uniquely determine the first. Even the informal statement "f is a constant function" may reasonably be formalized by requiring all values of f to agree, by asserting the existence of a common value, or by identifying f with a constant function. Whether such formulations are formally interchangeable can depend on implicit domain assumptions and on principles available in the ambient theory. A reference-based evaluator must therefore make explicit which distinctions it preserves and which variations it treats as equivalent.

We therefore view reference-based autoformalization evaluation as the selection and certification of an appropriate formal equivalence relation. Syntactic equivalence preserves almost every representational choice and rejects many harmless reformulations, whereas whole-statement logical equivalence can identify statements with unrelated mathematical content. For statement-level autoformalization, we introduce *Structure-Aware Equivalence* (SAE), an intermediate relation designed to preserve translation-relevant mathematical roles while allowing formally justified reformulation within those roles. For statements with assumptions and a conclusion, SAE preserves this premise–conclusion organization; for relational statements, it preserves the mathematical expressions and relations being asserted. Across both cases, it maintains a consistent correspondence between the mathematical objects of the two statements. SAE is not intended as a universal definition of mathematical equivalence, but as one conservative, role-preserving point in an explicit hierarchy of possible criteria.

We operationalize SAE through the *Autoformalization Correctness Evaluator* (ACE). SAE specifies *what* relation should be checked; ACE implements the corresponding comparison and uses Lean-backed verification to certify its outcome. ACE reports *Certified SAE-Equivalent* when a certified obligation establishes the required SAE conditions, and *Certified SAE-Inequivalent* only when every admissible alternative is ruled out. If any required comparison remains unresolved by proof search, ACE reports *Inconclusive* rather than converting failed proof search into a claim of inequivalence. Proof-search strength therefore affects certification coverage only, not the evaluation criterion.

We evaluate ACE on the general-purpose ProofNetVerif benchmark (Poiroux et al., 2025) and two targeted diagnostic sets. On ProofNetVerif, a general-purpose benchmark introduced with BEq+, ACE shows competitive coverage on consensus-equivalent pairs while remaining highly conservative on disagreement-equivalent and inequivalent pairs. The diagnostic sets expose complementary failure modes of existing approaches: BEq+ accepts 94–96% of unrelated but whole-statement logically equivalent pairs, and LeanScorer accepts 42% of localized meaning-changing perturbations; ACE accepts none of either set as equivalent. These results separate the appropriateness of the correctness criterion from the coverage of the procedure used to certify it.

Our contributions are threefold:

- We introduce *Structure-Aware Equivalence* (SAE), a role-preserving criterion between syntactic and whole-statement logical equivalence.
- We develop *ACE*, which operationalizes SAE with formally certified equivalence and inequivalence judgments and an explicit Inconclusive outcome.
- We evaluate SAE and ACE on an existing human-annotated benchmark introduced with BEq+ and targeted diagnostics, exposing complementary failure modes of learned semantic scoring and whole-statement logical equivalence.

2 Problem Formulation and Structure-Aware Equivalence

This section defines *structure-aware equivalence*, motivated by the structure of natural-language mathematical statements. We specify statement normalization, admissible variable alignment, and comparison rules for premise-bearing and premise-free statements. This distinction reflects how we capture mathematical intent: through the separate roles of assumptions and conclusions in premise-bearing statements, and through the asserted relations and the expressions they connect in premise-free statements. We then relate the criterion to syntactic equivalence and logical equivalence and define the corresponding notion of inequivalence. Section 3 describes how ACE certifies these judgments through Lean proof obligations.

Normalized statement form. Following the quantifier-prefix representation of prenex normal form (Enderton, 2001), we use a restricted prenex-style normalization: $N(T) = Q_1(x_1 : A_1) \cdots Q_n(x_n : A_n).\ H_1 \to \cdots \to H_m \to C_T$, where $Q_i \in \{\forall, \exists\}$. Normalization uses equivalence-preserving transformations and records quantifier scope and binder dependencies; quantifiers that are not lifted remain inside their components. We write $\Delta_T = \big(Q_1(x_1 : A_1), \ldots, Q_n(x_n : A_n)\big)$ for the ordered quantifier prefix, $\Gamma_T = (H_1, \ldots, H_m)$ for the top-level premises, and C_T for the residual conclusion (Coquand & Huet, 1988; Harper et al., 1993). Each prefix binder records its quantifier kind and type, which may depend on earlier variables. When $m = 0$, the body is simply C_T, and we set $\bigwedge \Gamma_T = \text{True}$. For example, $\forall x : \mathbb{N},\ \exists y : \mathbb{N},\ x > 0 \to x < y$ has $\Delta_T = (\forall x : \mathbb{N},\ \exists y : \mathbb{N}), \Gamma_T = (x > 0), C_T = (x < y)$.

Normalization fixes the premise–conclusion boundary, which comparisons preserve. A statement is *premise-bearing* when $m > 0$ and *premise-free* when $m = 0$. For premise-bearing statements, the premise conjunction and conclusion are compared by proposition-level equivalence, including any quantifiers retained within these components. For premise-free statements, the conclusion must satisfy the grammar $C ::= E_1\, \rho\, E_2 \mid C \wedge C \mid C \vee C,\ \rho \in \{=, \neq, <, \leq, >, \geq\}$, where E_1 and E_2 are well-typed expressions. In both cases, prefix quantifiers are handled by admissible alignment. A statement is *supported* if the prescribed normalization produces a form satisfying these restrictions[1].

Admissible variable alignment. Comparing the roles of mathematical objects requires a consistent correspondence between the variables of the two statements. For two statements T and T', an *admissible alignment* π is a bijection between their extracted object variables: the term variables recorded in Δ_T and $\Delta_{T'}$, as distinct from the proof hypotheses recorded in their premise sequences. An alignment must satisfy four conditions: **(i) Binder correspondence:** every extracted object variable on either side has exactly one counterpart; **(ii) Type preservation:** after substituting corresponding variables, aligned binder types agree under type checking, and the induced substitutions are well typed in both directions; **(iii) Dependency preservation:** each binder follows all binders on which its type depends; and **(iv) Quantifier preservation:** aligned binders have the same quantifier kind, and reordering is restricted to independent binders within contiguous blocks of the same quantifier kind.

These conditions permit alpha-renaming (consistent renaming of bound variables) and well-typed reordering of independent binders within universal or existential blocks (de Bruijn, 1972; Barendregt, 1985). They preserve quantifier alternation and binder dependencies. Alignments act only on the extracted variables; they do not move binders across component boundaries or rearrange quantifiers retained inside components.

Let $\mathsf{Adm}(T, T')$ denote the finite set of admissible alignments. For an alignment π, let Δ_π be the shared local context containing the aligned variables and their types, and write Γ_T^π and C_T^π for the

[1] Over 99.9% of the data in the dataset we use are supported. See Section 4.1 for dataset details.

premises and conclusion of T expressed in this context, and similarly for T'. Quantifier kinds and scope are retained in the aligned prefixes when reconstructing the complete statements.

Premise-Conclusion equivalence. For premise-bearing natural language statements, readers typically distinguish the assumptions specifying when a claim applies from the conclusion specifying what follows. This distinction makes the roles of the components central to interpretation, while allowing their logical expressions to take different forms. We preserve this organization by comparing the premise conjunctions and conclusions separately under the same alignment, allowing logically equivalent reformulations within each role.

Definition 2.1 (Premise-Conclusion equivalence)**.** Two normalized premise-bearing statements satisfy $T \equiv_B T'$ if there exists $\pi \in \mathsf{Adm}(T, T')$ such that $\Delta_\pi \vdash (\bigwedge \Gamma_T^\pi) \leftrightarrow (\bigwedge \Gamma_{T'}^\pi)$ and $\Delta_\pi \vdash C_T^\pi \leftrightarrow C_{T'}^\pi$.

Here, $\Delta_\pi \vdash \Phi$ means that the proposition Φ is *provable* in the shared context Δ_π. Both obligations are proved without assuming either statement's top-level premises. Comparing premise conjunctions permits premise reordering and logical reformulation without requiring a positional match between individual hypotheses. For example, consider $T : \forall x, y : \mathbb{N},\ x > 0 \to x + y > 0$, $T' : \forall a, b : \mathbb{N},\ b > 0 \to a + b > 0$. The alignment $x \mapsto b,\ y \mapsto a$ gives the obligations $b > 0 \leftrightarrow b > 0$ and $b + a > 0 \leftrightarrow a + b > 0$. The first is immediate; the second follows from commutativity of addition.

The same alignment must establish both comparisons. Matching the premises under one variable correspondence and the conclusions under another is insufficient.

Requiring unconditional conclusion equivalence also excludes some transformations that preserve the complete implication. For example, $P \to Q$ and $P \to (P \wedge Q)$ are logically equivalent, but their conclusions Q and $P \wedge Q$ are not generally equivalent without assuming P. Likewise, contraposition alone does not establish premise-bearing equivalence between $P \to Q$ and $\neg Q \to \neg P$. These restrictions preserve the distinction between assumptions and conclusions.

Premise-free equivalence. For natural language statements expressing simple relations, human readers tend to assess equivalence by comparing the quantities involved and the relations asserted between them. Comparing only the truth of the complete conclusion would instead identify any two provable statements. We therefore retain the relational content by comparing the normalized conclusion trees and their relation operands under a single admissible alignment. At an internal node, both trees must use the same connective. For $\wedge$ and $\vee$, the children may be paired directly or in reversed order. Thus, matching preserves the normalized tree structure up to child swaps. The matching rule itself does not perform reassociation or distributive rewrites.

At a pair of relational leaves, $E_1\, \rho\, E_2$ and $E_1'\, \rho'\, E_2'$, one of the following orientations must succeed: **(i) Direct orientation:** $\rho = \rho'$, and $\Delta_\pi \vdash E_1 = E_1',\ \Delta_\pi \vdash E_2 = E_2'$; or **(ii) Reversed orientation:** $\rho' = \mathrm{dual}(\rho)$, and $\Delta_\pi \vdash E_1 = E_2',\ \Delta_\pi \vdash E_2 = E_1'$. All operands are expressed in the shared context, and each equality compares terms of the same type. Equality and disequality are self-dual; $<$ is dual to $>$, and $\leq$ is dual to $\geq$.

Definition 2.2 (Premise-free equivalence)**.** Two normalized premise-free statements satisfy $T \equiv_F T'$ if there exists $\pi \in \mathsf{Adm}(T, T')$ under which their conclusion trees admit a recursive matching satisfying every connective, relation, and operand-equality condition.

All leaves use the same variable alignment, while child order and relation orientation may be chosen independently at each matching node. The criterion is intentionally stricter than logical equivalence: over $\mathbb{R}$, $a < b$ and $a + 1 < b + 1$ are logically equivalent, but their corresponding operands are not equal under the identity alignment and therefore do not match.

Structure-aware equivalence. To reflect how humans interpret assumption–conclusion roles and relational content in natural language, we combine the premise-bearing and premise-free comparison rules into a single relation on supported statements.

Definition 2.3 (Structure-aware equivalence)**.** Two supported statements T, T' are *structure-aware equivalent*, written $T \equiv_{SA} T'$, if both are premise-bearing and $T \equiv_B T'$, or both are premise-free and $T \equiv_F T'$.

A supported pair with premises on only one side is therefore not structure-aware equivalent.

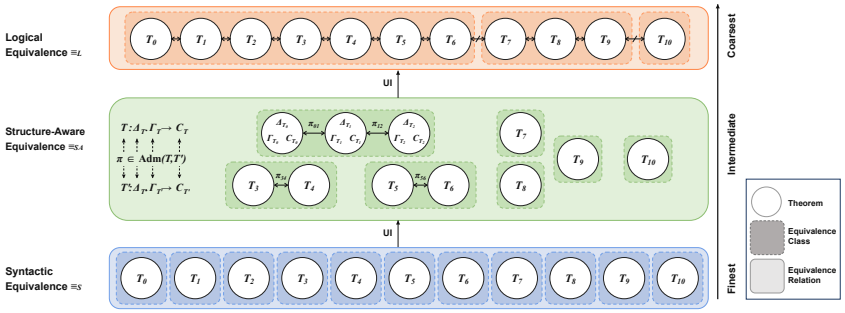


Figure 1: A hierarchy of equivalence relations including syntactic equivalence, structure-aware equivalence, and logical equivalence.

To connect this relation to verification, we construct a proposition $E(T, T')$ that collects the comparison obligations over all admissible alignments and structural matchings. A correspondence establishing structure-aware equivalence yields a proof of this proposition. For an admissible alignment π, let $\Delta_\pi = (z_1 : B_1, \ldots, z_k : B_k)$ be the shared context. For any proposition Φ whose free variables are contained in this context, write $\forall \Delta_\pi.\, \Phi \;:=\; \forall (z_1 : B_1) \cdots (z_k : B_k),\, \Phi$. This notation universally closes the comparison obligations. The original statements' quantifier kinds and scope remain governed by admissibility.

For *premise-bearing statements*, define $E_\pi := \forall \Delta_\pi.\, [(\bigwedge \Gamma_T^\pi \leftrightarrow \bigwedge \Gamma_{T'}^\pi) \wedge (C_T^\pi \leftrightarrow C_{T'}^\pi)]$. Both comparisons must hold without assuming either statement's top-level premises. For *premise-free statements*, consider all complete structural matchings permitted under π. Each matching preserves connectives up to the allowed child swaps and pairs relational leaves with compatible relations, operand types, and orientations. Represent each matching by a collection μ of aligned operand pairs whose equality it requires, and let $\mathcal{M}_\pi$ be the finite set of these collections. A direct match between $E_1\, \rho\, E_2$ and $E_1'\, \rho'\, E_2'$ contributes (E_1, E_1') and (E_2, E_2'); a reversed match contributes (E_1, E_2') and (E_2, E_1'). Define $E_\pi := \bigvee_{\mu \in \mathcal{M}_\pi} \left(\forall \Delta_\pi.\, \bigwedge_{(u,v)\in\mu} u = v \right)$.

The combined comparison proposition is

$$E(T, T') := \begin{cases} \bigvee_{\pi \in \mathsf{Adm}(T,T')} E_\pi, & \text{if both } T \text{ and } T' \text{ are premise-bearing or both are premise-free,} \\ \mathrm{False}, & \text{otherwise.} \end{cases}$$

The disjunctions over alignments and matchings occur outside the universal closure, requiring a correspondence that works uniformly across all aligned values.

By construction, $T \equiv_{SA} T' \Longrightarrow \vdash E(T, T')$, where $\vdash E(T, T')$ means that the comparison proposition is provable in the ambient formal system. Indeed, an alignment and matching witnessing structure-aware equivalence provide proofs of the required component comparisons; these combine into a proof of the corresponding alternative in $E(T, T')$.

The criterion depends on the normalized formulation: introducing or removing top-level premises can change the applicable comparison rule unless normalization removes that difference.

Equivalence hierarchy. Structure-aware equivalence lies between syntactic equivalence and logical equivalence. Let $\equiv_S$ denote syntactic equivalence of elaborated statements, and $\equiv_L$ denote whole-statement logical equivalence (in short, logical equivalence): $T \equiv_L T' \iff \vdash T \leftrightarrow T'$. On the supported fragment, under the fixed normalization, $T \equiv_S T' \Longrightarrow T \equiv_{SA} T' \Longrightarrow T \equiv_L T'$. The first implication follows from the identity alignment. The second follows because every SAE witness establishes equivalence of the normalized bodies uniformly.

Structure-aware equivalence is reflexive, symmetric, and transitive. These properties follow from identity, inversion, and composition of admissible alignments and structural matchings, together with the corresponding properties of the component comparisons. Figure 1 illustrates the resulting partitions. Syntactic equivalence gives the finest partition, SAE permits the specified changes in

formulation, and logical equivalence gives the coarsest partition. Premise-bearing and premise-free equivalence remain two cases within SAE, rather than separate levels of the hierarchy.

Finer intermediate relations are also possible. For example, one could recursively impose structure-aware comparison within premise-bearing components. Such a criterion would preserve additional structure at the cost of admitting fewer reformulations. We view SAE as one explicit point in this design space approximating the human intent, rather than the exact notion of translation equivalence.

Structure-aware inequivalence. Within the supported fragment, *structure-aware inequivalence*, written $T \not\equiv_{SA} T'$, is the complement of SAE. For two premise-bearing statements, inequivalence means that no admissible alignment satisfies both component comparisons. For two premise-free statements, it means that no admissible alignment admits a recursive matching satisfying all required conditions. A supported pair with top-level premises on only one side is also structure-aware inequivalent. In the unified formulation, $\neg E(T, T')$ rules out every admissible alternative.

3 ACE Verification Framework

ACE operationalizes SAE in two stages: normalizing and aligning each reference–candidate pair to generate Lean obligations, then applying proof search with Lean-checked proofs (Moura & Ullrich, 2021). The outcome is **Certified SAE-Equivalent**, **Certified SAE-Inequivalent**, or **Inconclusive**.

3.1 Normalization and Alignment

The pipeline begins by elaborating the theorem type and normalizing its quantifier structure into a canonical representation with a prenex-style quantifier prefix: an ordered sequence of $\forall$ and $\exists$ binders followed by a residual matrix. This representation preserves dependencies between binders and whether a binder is a typeclass instance, while retaining implications in the matrix rather than eliminating them. Lean then verifies that the reconstructed proposition elaborates successfully before structural comparison.

After validation, the pipeline enumerates admissible binder alignments to reconcile permitted presentation differences. These alignments must respect type, role, and dependency constraints and may match a matrix premise with a corresponding instance requirement when their normalized types are compatible. Their scope and limitations are summarized in Appendix A.

3.2 Proof Search and Verification

Using the normalized and aligned representations, ACE constructs proof obligations for the comparison proposition $E(T, T')$ defined in Section 2. It batches the positive obligations as a disjunction, so proving any one admissible branch suffices to establish $E(T, T')$. If the positive obligations do not establish equivalence, ACE constructs the corresponding negated branch obligations as a conjunction. A certified negative result therefore requires every represented branch to be rejected.

Each obligation is encoded as a synthetic Lean theorem and submitted to the selected proof-search backend. Candidate proofs are accepted only when Lean closes the corresponding proof state. ACE invokes a fixed baseline fallback tactic portfolio described in Appendix B for unsuccessful candidates. These verification results, together with the static structural checks, determine the final outcomes.

3.3 Verification and Outcomes

At the final decision stage, the static structural checks are combined with the verified obligation results. A pair that fails the structural conditions specified in Section 2 is classified directly as **Certified SAE-Inequivalent**; otherwise, the result depends on the above obligation verification.

Certified SAE-Equivalent is reported when at least one admissible alignment establishes all required equivalence obligations. **Certified SAE-Inequivalent** is reported only when every admissible alignment enumerated for the supported fragment is rejected. If any alignment remains unresolved, or if the complete candidate set cannot be enumerated or checked, the result is **Inconclusive**.

4 EVALUATION

To evaluate ACE, we assess its agreement with human intent on realistic reference–candidate pairs on ProofNetVerif, and its sensitivity against semantic perturbations using targeted diagnostic sets which keep provability or have similar surface forms. Because ProofNetVerif was introduced with BEq+ and predates SAE, it provides a general-purpose benchmark not tailored to our criterion.

4.1 DATASETS

4.1.1 PROOFNETVERIF

We use ProofNetVerif, introduced with BEq+ (Poiroux et al., 2025), as a general-purpose collection of reference–candidate pairs. Before evaluation, we exclude pairs whose Lean declarations fail to compile or whose formal statements do not elaborate to well-formed theorem propositions. The remaining well-formed pairs provide the raw source for our benchmark sets.

Equivalent Set. The *Equivalent Set* contains all pairs originally labeled *Equivalent* in ProofNetVerif. We re-audit this partition using a fixed criterion: can both formal statements be faithfully described by the same natural-language mathematical statement, allowing notational and standard definitional variation? Evaluator outputs do not determine the audit labels, which characterize agreement with the original annotations under this criterion.

We classify 787 pairs as the *Consensus-Equivalent Subset*: both formal statements admit the same faithful natural-language description, allowing notational, binder, and standard definitional variation. The remaining 340 pairs form the *Disagreement-Equivalent Subset*: their original label is unsupported under this criterion because they change assumptions, logical form, quantifiers, domains, or conclusion details. Mathematical interderivability alone is insufficient; an implication and a biconditional may therefore receive different classifications. For example, reference $\forall n : \mathbb{N},\ \mathrm{Odd}(n) \to 8 \mid (n^2-1)$ is different from $\forall n : \mathbb{Z},\ n \bmod 2 = 1 \to (n^2-1) \bmod 8 = 0$ by a change of the quantified domain, falling in the disagreement-equivalent subset. Additional auditing details and examples are provided in Appendix D.

Inequivalent Set. The *Inequivalent Set* consists of pairs carrying ProofNetVerif's original *Inequivalent* label and directly uses its 2,236 pairs. Unlike the audited partition above, these pairs generally involve substantive changes to the mathematical claim, such as reversed implications, changed assumptions, or different target propositions. Representative cases are provided in Appendix D.3.

4.1.2 DIAGNOSTIC SETS

To isolate systematic failure modes that may be diluted in the general-purpose benchmark, we construct two targeted diagnostic sets.

Unrelated Shared-Provability Set. Logical equivalence can hold between statements about unrelated mathematical objects or roles: a biconditional may be provable simply because both statements are independently provable. We ask GPT-5.6 Luna to generate such candidate pairs and, after manual review, retain 50 well-typed pairs for which Lean proves $T_r \leftrightarrow T_c$. This set tests whether biconditional proofs can ignore correspondence between binders, premises, conclusions, and mathematical objects. For example, $n \cdot 1 = n$ for $n : \mathbb{N}$ and $s = s$ for an arbitrary string s are both provable but concern unrelated objects and operations. Additional examples are provided in Appendix E.

Semantic Perturbation Set. Localized changes to premises, typeclass assumptions, numerical values, or conclusions can alter meaning while preserving most of a statement's surface form. We sample reference statements from ProofNetVerif and ask GPT-5.6 Luna to propose constrained perturbations; after manual review, we retain 50 pairs with changed meaning. The set tests whether reference-free semantic scoring overlooks such changes when unchanged components dominate the aggregate score. For example, one pair strengthens the typeclass requirement from $[\mathrm{Group}\ G]$ to $[\mathrm{CommGroup}\ G]$ while leaving the conclusion unchanged. See additional examples in Appendix F.

Table 1: Decision distributions on the ProofNetVerif subsets. Outcome labels follow Section 4.2.

Dataset		Consensus-Equivalent			Disagreement-Equivalent			Inequivalent		
Evaluator	Configuration	Outcome			Outcome			Outcome		
		Eq	Noneq	Inc	Eq	Noneq	Inc	Eq	Noneq	Inc
ACE	Aesop	55.4%	29.4%	15.2%	0.3%	79.4%	20.3%	0.1%	87.2%	12.7%
	Aristotle	56.9%	29.4%	13.7%	0.3%	79.4%	20.3%	0.1%	87.3%	12.6%
	GPT-5.6 Luna	60.0%	30.5%	9.5%	0.6%	82.1%	17.4%	0.3%	89.1%	10.6%
BEq+	Default	60.7%	39.3%*	0%	2.4%	97.6%*	0%	0.1%	99.9%*	0%
	Aesop	59.0%	41.0%*	0%	1.8%	98.2%*	0%	0.2%	99.8%*	0%
	GPT-5.6 Luna	68.9%	31.1%*	0%	4.4%	95.6%*	0%	0.7%	99.3%*	0%
LeanScorer	GPT-5.6 Luna	81.4%*	18.6%*	0%	43.8%*	56.2%*	0%	5.1%*	94.9%*	0%

Note. Default denotes BEq+'s original configuration, which applies a predefined sequence of Lean tactics. * These outcomes are not supported by formal certificates. BEq+ *Noneq* outcomes do not certify inequivalence and may reflect failed proof search. LeanScorer provides no formal certificate for either outcome.

4.2 Evaluation Setup

Evaluators and Provers. We compare three complementary approaches: ACE, a reference-based structure-aware evaluator; BEq+ (Poiroux et al., 2025), a reference-based logical-equivalence evaluator; and Mathesis LeanScorer (Yu et al., 2026), a reference-free semantic-scoring framework. For ACE, we use Aesop (Limperg & From, 2023) as an automated Lean proof searcher and Aristotle (Achim et al., 2025) and GPT-5.6 Luna (OpenAI, 2026) as LLM-based proof-search backends to discharge the resulting obligations. For BEq+, we evaluate its default configuration, based on a predefined Lean-tactic sequence, and augment it with Aesop and GPT-5.6 Luna for extended proof-search comparison. LeanScorer uses GPT-5.6 Luna to generate component-level semantic scores, which it aggregates into a final judgment.

Outcome Labels. For comparability, the tables use the labels *Eq*, *Noneq*, and *Inc*. For ACE, these denote *Certified SAE-Equivalent*, *Certified SAE-Inequivalent*, and *Inconclusive*, respectively. BEq+ reports only the *Eq* and *Noneq* outcomes and does not report *Inc*. Its *Noneq* outcome does not certify inequivalence and may reflect failed proof search (Poiroux et al., 2025). LeanScorer likewise reports only the *Eq* and *Noneq* outcomes, without a formal certificate for either outcome (Yu et al., 2026). Relative to the subset labels, these outcomes may be false positives or false negatives.

4.3 Results and Analysis

We present the overall results of the above experiments and address the following research questions.

RQ1: How does ACE behave on a general-purpose human-annotated benchmark? On the *Consensus-Equivalent* subset, ACE reports *Eq* for 55.4–60.0% of pairs across its three proof-search backends. Most certified *Noneq* cases are faithful reformulations accepted by the audit but excluded by SAE's conservative alignment policy, for example because binder, quantifier, or instance roles cannot be aligned. These cases reflect a difference between the broader notion of equivalence in the audit and the stricter role-preserving criterion in SAE. Representative cases are in Appendix G.

On the *Disagreement-Equivalent* subset, ACE reports *Noneq* for 79.4–82.1% of pairs and *Eq* for only 0.3–0.6%. The rare Eq cases mainly involve redundant premises or theorem-supported reformulations for which SAE and the audit draw the equivalence boundary differently. Similarly, on the original ProofNetVerif *Inequivalent* set, only 0.1–0.3% of pairs receive an *Eq* outcome; manual inspection attributes these cases to redundant premises, implicit-binder restoration, or alternative theorem-level reformulations. Representative analyses are provided in Appendix H.

RQ2: Does ACE preserve mathematical roles? On the *Unrelated Shared-Provability Set*, ACE reports *Noneq* across all three proof-search backends, yielding a 100% true-negative rate. Thus, ACE rejects pairs whose complete propositions are logically equivalent but whose mathematical roles do

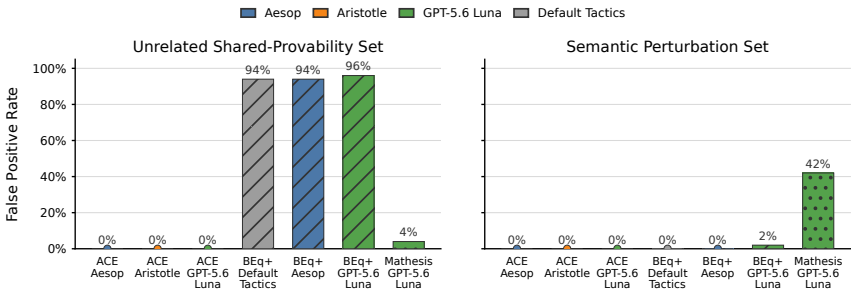


Figure 2: False-positive rates on the two targeted diagnostic sets. Each panel shows all reported evaluator configurations; colors identify the proof-search model, while the x-axis labels identify the evaluator and configuration. Zero rates are shown by markers at the baseline.

not correspond. By contrast, BEq+ reports *Eq* for 94–96% of the pairs across its configurations, while LeanScorer reports *Eq* for 4% (Figure 2).

This contrast reflects a mismatch between logical equivalence and role preservation. BEq+ verifies a biconditional between the complete propositions but does not require correspondence between their binders, premises, conclusions, or mathematical objects. It can therefore report *Eq* even when the candidate does not preserve the reference's intended mathematical roles.

The same criterion mismatch appears on the general-purpose benchmark. On ProofNetVerif's *Inequivalent* set, BEq+ reports *Eq* for 2 pairs with its Default configuration, 4 with Aesop, and 16 with GPT-5.6 Luna (Table 1). Counterintuitively, stronger proof search increases BEq+'s false-positive rate without improving agreement with human intent.

RQ3: Is ACE sensitive to localized semantic changes? On the *Semantic Perturbation Set*, ACE produces no *Eq* outcomes with any proof-search backend, indicating that it does not treat the localized changes as equivalent. It reports *Noneq* for 72–74% of the cases (Aesop: 72%, Aristotle: 74%, and GPT-5.6 Luna: 74%); the remaining 26–28% are *Inc* because bounded proof search does not resolve the corresponding obligations. These unresolved cases reflect verification coverage rather than insensitivity of the SAE criterion. A stronger proof-search backend may reduce this fraction without changing the criterion.

By contrast, LeanScorer reports a substantially higher *Eq* rate (Figure 2). Its reference-free aggregate score can allow unchanged components to outweigh a localized mismatch, revealing a limitation of semantic scoring for detecting semantic changes and reflecting human intent.

5 Limitations

ACE is limited by both the scope of SAE and bounded verification. SAE applies only to statements supporting the prescribed normalization and relational decomposition; its premise–conclusion boundary may reject valid reformulations, and its premise-free branch excludes arbitrary higher-order predicates and logical structure. Its guarantees depend on correct normalization and complete alignment enumeration. Incomplete, resource-bounded Lean proof search may leave supported obligations unresolved, yielding *Inconclusive* outcomes. SAE also fixes a particular granularity of structure preservation; stricter recursive variants are possible within the same framework.

6 Conclusion

We introduced SAE and ACE for reference-based evaluation of autoformalized mathematical statements. SAE preserves binder correspondence, premise–conclusion roles, and relational structure, while ACE provides Lean-checkable equivalence and inequivalence judgments with an explicit Inconclusive outcome. Experiments on ProofNetVerif and two targeted diagnostic sets show that this structure-aware criterion avoids complementary failure modes of whole-statement logical equivalence and semantic scoring while making proof-search limitations explicit.

AI USE STATEMENT

In this work, we used GPT-5.6 Luna to generate candidate pairs for the diagnostic datasets, and Aristotle and GPT-5.6 Luna to propose proof-search candidates used in the evaluation. We used GPT-6 Astra with high reasoning effort to assist with the initial audit and classification of ProofNetVerif cases. The authors reviewed these AI-assisted classifications against the formal statements and their normalized representations. We also used generative AI tools to assist with polishing the paper's writing. All AI-assisted outputs were reviewed by the authors, who take responsibility for the final content, claims, code, and datasets.

REFERENCES

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, Martin Michelsen, Riley Patterson, Eric Rodriguez, Laura Scharff, Vikram Shanker, Vladmir Sicca, Hari Sowrirajan, Aidan Swope, Matyas Tamas, Vlad Tenev, Jonathan Thomm, Harold Williams, and Lawrence Wu. Aristotle: Imo-level automated theorem proving, 2025. URL https://arxiv.org/abs/2510.01346.

Anthropic. Formalizing fermat's last theorem. Anthropic Research, September 2026. URL https://www.anthropic.com/research/formalizing-fermats-last-theorem.

Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics. *ArXiv*, abs/2302.12433, 2023. URL https://arxiv.org/abs/2302.12433.

Henk P. Barendregt. The lambda calculus: Its syntax and semantics. 1985. URL https://doi.org/10.2307/2274112.

Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inf. Comput.*, 76:95–120, 1988. URL https://doi.org/10.1016/0890-5401(88)90005-3.

N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Studies in logic and the foundations of mathematics*, 133:375–388, 1972. URL https://doi.org/10.1016/S0049-237X(08)70216-7.

Herbert B. Enderton. *A Mathematical Introduction to Logic*. Academic Press, 2 edition, 2001. ISBN 9780122384523. doi: 10.1016/C2009-0-22107-6. URL https://www.sciencedirect.com/book/9780122384523/a-mathematical-introduction-to-logic.

Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993. ISSN 0004-5411. doi: 10.1145/138027.138060. URL https://doi.org/10.1145/138027.138060.

Jannis Limperg and Asta Halkjær From. Aesop: White-box best-first proof search for lean. In *Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2023, pp. 253–266, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700262. doi: 10.1145/3573105.3575671. URL https://doi.org/10.1145/3573105.3575671.

Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 86205–86233, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/d630537fc4402cfa3ebbc7450a0cac91-Paper-Conference.pdf.

Jianqiao Lu, Yingjia Wan, Yinya Huang, Jing Xiong, Zhengying Liu, and Zhijiang Guo. Formalalign: Automated alignment evaluation for autoformalization. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Learning Representations*, volume 2025, pp. 102122–102149, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/fceedf8c9c0ff51f41b9fe0294ef0070-Paper-Conference.pdf.

Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction – CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings*, pp. 625–635, Berlin, Heidelberg, 2021. Springer-Verlag. ISBN 978-3-030-79875-8. doi: 10.1007/978-3-030-79876-5_37. URL https://doi.org/10.1007/978-3-030-79876-5_37.

OpenAI. Gpt-5.6 luna model. OpenAI API documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-luna.

Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Reliable evaluation and benchmarks for statement autoformalization. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17947–17969, Suzhou, China, nov 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.907. URL https://aclanthology.org/2025.emnlp-main.907/.

Hui Wei, Shenghua He, Tian Xia, Fei Liu, Andy Wong, Jingyang Lin, and Mei Han. Systematic evaluation of LLM-as-a-judge in LLM alignment tasks: Explainable metrics and diverse prompt templates. In *ICLR 2025 Workshop on Building Trust in Language Models and Applications*, 2025. URL https://openreview.net/forum?id=CAgBCSt8gL.

Ke Weng, Lun Du, Sirui Li, Wangyue Lu, Haozhe Sun, Hengyu Liu, and Tiancheng Zhang. Autoformalization in the era of large language models: A survey. May 2025. doi: 10.48550/arXiv.2505.23486. URL https://arxiv.org/abs/2505.23486.

Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. Autoformalization with large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NeurIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Qiming Xie, Zengzhi Wang, Yi Feng, and Rui Xia. Ask again, then fail: Large language models' vacillations in judgment. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10709–10745, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.577. URL https://aclanthology.org/2024.acl-long.577/.

Xuejun Yu, Jianyuan Zhong, Zijin Feng, Pengyi Zhai, Roozbeh Mohit, Wei Ng, Haoxiong Liu, Ziyi Shou, Jing Xiong, Yudong Zhou, Claudia Ong, Austen Sugiarto, Yaoxi Zhang, Wai Ming Tai, Huan Cao, Dongcai Lu, Jiacheng Sun, Qiang Xu, Shen Xin, and Zhenguo Li. Mathesis: Towards formal theorem proving from natural languages. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 91535–91559, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/93d5797656d3e70dca7535160a1458f3-Paper-Conference.pdf.

A Presentation Alignment Boundaries

This appendix specifies the implementation boundaries of the presentation-level alignment described in Section 3.1. The procedure aligns restricted Lean presentations before the formal SAE comparison; it does not introduce additional semantic equivalence rules.

A.1 Scope and Preserved Roles

Instance binders remain instance binders because they affect Lean elaboration and typeclass synthesis. Ordinary proposition-valued hypotheses used by the conclusion remain explicit prefix binders; hypotheses not used by the conclusion may instead appear as matrix implications. Presentation alignment may match such a matrix premise with a corresponding instance requirement, but it does not globally convert one binder role into another.

A.2 Permitted Presentation Alignments

Requirement matching uses a restricted normalized syntactic key that ignores superficial whitespace, parentheses, and local binder or universe names. Matching does not invoke Lean definitional equality, typeclass search, theorem proving, or semantic aliases.

For a matrix-premise alignment, ACE considers only consecutive leading top-level implications and counterpart instance requirements before the first existential binder. A premise is lifted to the counterpart's instance position only when the normalized requirement types and binder roles match; each instance requirement can be used at most once. The scan stops at the first unmatched premise and does not inspect nested implications. A subsequent named-prefix pass may align an ordinary named binder with one parsed argument to a remaining instance requirement under the same key. Binders after the first existential, incompatible roles or types, and many-to-one matches are outside this presentation-alignment procedure.

A.3 Subsequent Binder Alignment and Failure

After presentation alignment, the evaluator enumerates route-specific admissible binder alignments subject to type, role, and dependency constraints. The resulting alignment is shared by the complete statement, including its premise and conclusion. Presentation alignment alone does not decide equivalence; it returns an unchanged or aligned representation for the subsequent SAE comparison. If no safe shared binder alignment can be constructed, the evaluator applies the structural SAE decision rule without constructing an ill-typed Lean obligation.

B Fallback Tactic Portfolio

When a backend-generated tactic does not close a Lean obligation, ACE invokes the fixed fallback tactic portfolio. The portfolio is tried sequentially under the same Lean proof budget. It begins with reflexivity and targeted algebraic or logical normalization, including `rfl`, `ac_rfl`, `simpa`, `norm_num`, `ring_nf`, `linarith`, and `omega`. It then tries structural and propositional tactics such as `constructor`, `intro`, `exact`, `tauto`, `simp`, `aesop`, and `nlinarith`, followed by `ring`, `field_simp`, `simp_all`, and `funext` variants. The rewrite lemmas used by these tactics include logical commutativity and associativity, relation converses, and associative or commutative addition and multiplication: `eq_comm`, `ne_comm`, `and_assoc`, `and_left_comm`, `and_comm`, `or_assoc`, `or_left_comm`, `or_comm`, `iff_comm`, `ge_iff_le`, `gt_iff_lt`, `add_assoc`, `add_left_comm`, `add_comm`, `mul_assoc`, `mul_left_comm`, and `mul_comm`, together with the corresponding modular-arithmetic commutativity lemmas such as `Int.modEq_comm` and `Nat.ModEq.comm`.

For the premise-free structural cross-check, the same fallback portfolio is used for operand-level proof obligations. If Lean reports missing variables or an initialization error while constructing such an obligation, the evaluator may reconstruct it with the inferred implicit variables and retry up to three times after the initial attempt. These recovery retries do not change the comparison criterion; they only make the generated Lean obligation well-formed before proof search continues.

C Evaluation Resource Budgets

ACE uses a 600-second per-item budget, a 600-second budget for each synthetic Lean obligation, and a 60-second timeout for each individual tactic. Alignment branches are grouped into batches of at most four. Aesop supplies up to 20 deterministic tactic candidates; Aristotle and GPT-5.6 Luna each supply one generated candidate by default; each generated candidate is followed, when needed, by the fixed baseline fallback tactic portfolio. Transient Lean REPL initialization is retried up to twice, and a terminated reusable REPL is restarted once. After the generated candidates have been tried, ACE does not generate additional candidates after Lean rejects them; it proceeds with the fixed baseline fallback portfolio.

Table 2: Reason summary for the 340 pairs in the Disagreement-Equivalent subset. The categories are mutually exclusive and exhaustive.

Reason category	Count	Interpretation
Conclusion or predicate	134	The asserted predicate or conclusion changes materially.
Domain or structure	111	The mathematical domain, type, or structure changes.
Assumptions	52	A substantive hypothesis or required condition changes.
Quantifiers or witnesses	26	Quantifier scope, witness dependence, or related logical structure changes.
Logical form	17	The logical form changes in a way that affects the mathematical obligation.
Total	**340**	

D PROOFNETVERIF CASES

This appendix documents the manual audit of ProofNetVerif's original Equivalent partition and provides representative cases for the Consensus-Equivalent, Disagreement-Equivalent, and Inequivalent classes. The table summarizes the reasons assigned to the 340 Disagreement-Equivalent pairs used in the primary analysis.

Audit Procedure. We used GPT-6 Astra with high reasoning effort to organize the reference–candidate pairs and propose initial classifications with supporting rationales. Human review then checked each proposal against the original Lean declarations and normalized statements, focusing on domains, assumptions, quantifier scope, formal roles, mathematical objects, operations, and conclusions. Boundary cases were resolved under the classification rules defined in this appendix. The final audit labels were assigned independently of ACE's evaluator outcomes; whole-proposition provability alone was not treated as evidence of faithful equivalence.

D.1 CONSENSUS-EQUIVALENT SUBSET CASES

Classification rule. The audit retains pairs in the Consensus-Equivalent subset when the reference and candidate preserve the same mathematical specification. This includes renaming and formatting changes; explicit versus implicit parameters; elementary logical rewrites such as $\neg P$ versus $P \to \mathrm{False}$, reordered conjunctions, and curried hypotheses; function equality versus pointwise equality; equivalent finiteness presentations such as $\mathrm{Finite}\ G$ versus $\mathrm{Fintype}\ G$; and standard equivalent predicates on the same mathematical objects. An assumption may also be repackaged or added when it is already entailed by the existing assumptions or has a recoverable witness, provided that the admissible domain and mathematical obligation are unchanged.

Representative cases. *Implicit binder versus explicit quantification.*

```
Reference:
theorem reference_statement {n : ℕ} (hn : Odd n) :
  8 ∣ (n ^ 2 - 1)

Candidate:
theorem dummy :
  ∀ n : ℕ, Odd n → 8 ∣ (n ^ 2 - 1)
```

The candidate makes the implicit binder and premise explicit without changing the mathematical claim.

Reversed equality.

```
Reference:
theorem exercise_1_1_22a {G : Type*} [Group G] (x g : G) :
  orderOf x = orderOf (g⁻¹ * x * g)

Candidate:
theorem dummy {G : Type*} [Group G] (x g : G) :
  orderOf (g⁻¹ * x * g) = orderOf x
```

The candidate reverses the two sides of an equality without changing the mathematical claim.

Equivalent cardinality notation.

```
Reference:
theorem exercise_2_1_21 (G : Type*) [Group G] [Fintype G]
  (hG : card G = 5) : ∀ a b : G, a * b = b * a

Candidate:
theorem dummy {G : Type*} [Group G] [Fintype G]
  (hG : Fintype.card G = 5) : ∀ a b : G, a * b = b * a
```

The two statements express the same group-cardinality condition and the same commutativity conclusion; only the declaration style and cardinality notation differ.

Explicit versus implicit complex binders.

```
Reference:
theorem reference_statement (x y : ℂ) :
  |(|x|) - (|y|)| ≤ |x - y|

Candidate:
theorem dummy {x y : ℂ} :
  ||x| - |y|| ≤ |x - y|
```

The candidate changes only the binder explicitness and superficial notation; the mathematical inequality is unchanged.

D.2 Disagreement-Equivalent Subset Cases

Classification rule. The audit assigns pairs to the Disagreement-Equivalent subset when the reference and candidate do not preserve the same specification, including changes to quantified domains or algebraic structures (for example, $\mathbb{N}$ versus $\mathbb{Z}$ or an arbitrary field versus $\mathbb{R}$), changes from an arbitrary structure to a particular model, substantive assumptions or restrictions, changed existential or universal scope or witness dependence, and changes to mathematical objects, operations, predicates, or conclusions. Examples include replacing union by span, convergence by summability, or an implication by a non-equivalent biconditional. Whole-proposition provability or logical interderivability alone is not sufficient for the Consensus-Equivalent label. The cases below are representative examples of these categories, not an exhaustive list.

Representative cases.

```
Reference:
theorem reference_statement {R : Type*} [Ring R]
  (h : ∀ x : R, x ^ 3 = x) : Nonempty (CommRing R)

Candidate:
theorem dummy (R : Type*) [Ring R]
  (h : ∀ x : R, x ^ 3 = x) : ∀ a b : R, a * b = b * a
```

The reference asserts the existence of a commutative-ring structure on R, whereas the candidate asserts commutativity of the multiplication supplied by the existing ring structure. These are different mathematical obligations.

```
Reference:
theorem reference_statement {K V : Type*} [Field K] [AddCommGroup V]
  [Module K V] {S : Set V} (hS : Set.Countable S)
  (hS1 : span K S = ⊤) {ι : Type*} (R : ι → V)
  (hR : LinearIndependent K R) : Countable ι

Candidate:
theorem dummy {V : Type*} [AddCommGroup V] [Module ℝ V]
  {S : Set V} (hS : Set.Countable S)
  (hS_span : Submodule.span ℝ S = ⊤) :
  ∀ T : Set V, LinearIndependent ℝ
    (Subtype.val : T → V) → T.Finite ∨ Set.Countable T
```

The reference ranges over an arbitrary field, scalar module, and indexed family, while the candidate specializes to real vector spaces and subsets. The domains and quantified objects therefore differ substantially.

D.3 Inequivalent Set Cases

The Inequivalent Set contains substantive changes to the stated mathematical obligation, including reversed implications, changed quantifier scope or witness dependence, altered domains or types, different mathematical objects or operations, and changed target propositions. The following examples are from the original ProofNetVerif Inequivalent partition.

Reversed implication.

```
Reference:
theorem reference_statement {G H : Type*} [Group G] [Group H]
  (h : IsSolvable G) (hH : H ≤ G) : IsSolvable H

Candidate:
theorem dummy {G H : Type*} [Group G] [Group H]
  (h : IsSolvable H) (hH : H ≤ G) : IsSolvable G
```

The candidate reverses the subgroup implication, changing the mathematical claim.

Changed quantifier scope and square representations.

```
Reference:
theorem reference_statement :
  ∀ N : ℕ, ∃ n : ℕ, n > N ∧ ∃ i : Fin 6 → ℕ,
    n = (i 0)^2 + (i 1)^2 ∧
    n + 1 = (i 2)^2 + (i 3)^2 ∧
    n + 2 = (i 4)^2 + (i 5)^2

Candidate:
theorem dummy (n : ℕ) :
  ∃ m : ℕ, m > n ∧ ∃ (a b c : ℕ),
    m = a ^ 2 + b ^ 2 ∧
    m + 1 = a ^ 2 + c ^ 2 ∧
    m + 2 = b ^ 2 + c ^ 2
```

The reference uses a universally quantified bound and six potentially different square components. The candidate changes the quantifier structure and reuses only three components, so it states a different claim.

E Unrelated Shared-Provability Cases

Systematic failure pattern. This diagnostic set covers pairs whose complete propositions are both provable, or otherwise admit a whole-proposition equivalence proof, while their binders, premises, conclusions, mathematical objects, or operations have no faithful correspondence. The set therefore targets the general failure mode in which whole-proposition provability is mistaken for preservation of mathematical roles.

```
Reference:
theorem beq_fp_arithmetic_add_zero_vs_set_identity (n : Nat) :
  n + 0 = n

Candidate:
theorem beq_fp_arithmetic_add_zero_vs_set_identity (s : Set Int) :
  s = s
```

Manual review labels the pair as inequivalent, while BEq+ accepts it. ACE rejects the pair because the binder types and mathematical objects cannot be aligned.

Negation of false versus natural-number reflexivity.

```
Reference:
theorem reference_statement : Not False

Candidate:
theorem dummy : (37 : Nat) = 37
```

Both propositions are provable, but the reference states the negation of falsehood while the candidate states reflexivity of a natural number. Their mathematical roles and objects are unrelated.

F SEMANTIC PERTURBATION CASES

Systematic failure pattern. This diagnostic set covers localized changes to typeclass assumptions, numerical values, premises, conclusions, and other components while preserving most of the surrounding statement. These changes can alter the admissible algebraic structures or the asserted mathematical obligation without producing a large surface-level difference. We focus here on the two typeclass perturbations shown below.

Stronger typeclass requirement.

```
Reference:
theorem exercise_2_4_4 {G : Type*} [Group G] (H : Subgroup G) :
  closure ((H : Set G) \ {1}) = H

Candidate:
theorem exercise_2_4_4 {G : Type*} [CommGroup G]
  (H : Subgroup G) :
  closure ((H : Set G) \ {1}) = H
```

The candidate imposes the stronger commutativity requirement while leaving the visible conclusion unchanged. This changes the class of admissible structures and is therefore a semantic perturbation rather than a presentation-only difference.

Weaker typeclass requirement.

```
Reference:
theorem exercise_7_1_11 {R : Type*} [CommRing R] [IsDomain R]
  {x : R} (hx : x^2 = 1) : x = 1 ∨ x = -1

Candidate:
theorem exercise_7_1_11 {R : Type*} [Ring R] [IsDomain R]
  {x : R} (hx : x^2 = 1) : x = 1 ∨ x = -1
```

Here the candidate weakens the commutativity requirement from a commutative ring to a ring. Although the conclusion has the same surface form, the statement ranges over a different class of algebraic structures.

G ACE–AUDIT DISAGREEMENTS ON CONSENSUS-EQUIVALENT CASES

The Consensus-Equivalent false negatives arise when the audit accepts a faithful reformulation but ACE cannot construct a permitted shared alignment. The relevant forms include explicit versus implicit binders, elementary logical rewrites, equivalent predicate presentations, different packaging of the same assumptions, and standard definition unfoldings. After normalization, these changes may expose different binder counts, types, quantifier roles, or dependencies. The examples below illustrate two of these forms.

Conjunctive hypotheses and intersection notation.

```
Reference:
theorem reference_statement (X : Type*) [MetricSpace X] [CompleteSpace X]
  [Nonempty X] (G : ℕ → Set X)
  (hG : ∀ n, IsOpen (G n) ∧ Dense (G n)) :
  ∃ x, ∀ n, x ∈ G n

Candidate:
theorem dummy {X : Type*} [MetricSpace X] [CompleteSpace X]
  [Nonempty X] (G : ℕ → Set X)
  (hG_{open} : ∀ n, IsOpen (G n))
  (hG_{dense} : ∀ n, Dense (G n)) :
  (⋂ n, G n).Nonempty
```

The candidate separates the conjunction into two hypotheses and writes the existential conclusion as nonemptiness of an intersection. The mathematical claim is unchanged, but the normalization exposes a different prefix and therefore no permitted shared binder alignment is found.

Named predicate and defining condition.

```
Reference:
```

```
theorem reference_statement {R : Type*} [Ring R] {x : R}
  (hx : IsNilpotent x) : IsUnit (1 + x)

Candidate:
theorem dummy {R : Type*} [Ring R] {x : R}
  (hx : ∃ n : ℕ, x ^ n = 0) : IsUnit (1 + x)
```

The candidate unfolds the standard predicate `IsNilpotent x` into its defining existential condition. The mathematical claim is preserved, but the unfolded witness becomes an additional canonical binder, so ACE conservatively rejects the pair at the alignment stage.

H ACE–AUDIT DISAGREEMENTS ON DISAGREEMENT-EQUIVALENT CASES

This section gives representative cases in which ACE returns *Eq* despite the audited or original dataset label being inequivalent. It covers two sources of such outcomes: cases assigned to the Disagreement-Equivalent subset by the audit, and cases carrying the original ProofNetVerif Inequivalent label.

These false positives arise at the boundary between the audit and the certified comparison. They include explicit added conditions that ACE can discharge from existing assumptions, equivalent-looking predicates or theorem-level reformulations supported by Lean, and other formal differences that the audit does not treat as mere presentation. The following examples illustrate these forms.

```
Reference:
theorem reference_statement {G : Type*} [Fintype G] [Group G]
  (hG : card G = 132) : ¬ IsSimpleGroup G

Candidate:
theorem dummy (G : Type*) [Group G] [Fintype G]
  (hG : Fintype.card G = 132) (h' : 1 < Fintype.card G) :
  ¬ IsSimpleGroup G
```

The added cardinality premise follows from $\mathrm{card}(G) = 132$ and is therefore redundant. The audit nevertheless treats the explicit addition as a formal statement difference, whereas ACE can prove the resulting comparison obligation.

```
Reference:
theorem reference_statement {X : Type*} [TopologicalSpace X]
  (hX : LocallyCompactSpace X) (hX' : T2Space X) :
  RegularSpace X

Candidate:
theorem dummy {X : Type*} [TopologicalSpace X]
  [H : LocallyCompactSpace X] [T2 : T2Space X] : T3Space X
```

The candidate names a stronger separation predicate rather than the reference predicate. Under the Hausdorff assumption, existing Lean theorems make the two obligations interderivable, so ACE proves equivalence even though the audit distinguishes the target predicates.

ACE also returns *Eq* for a small number of pairs carrying the original *Inequivalent* label. Manual review shows that these are apparent false positives relative to that benchmark label rather than genuine changes in mathematical content. The recurring forms include redundant premises, omitted variables restored as implicit binders during elaboration, and alternative theorem-level formulations whose equivalence follows from existing Lean theorems.

Redundant premises.

```
Reference:
theorem reference_statement {G : Type*} [Fintype G] [Group G]
  (hG : card G = 1365) : ¬ IsSimpleGroup G

Candidate:
theorem dummy {G : Type _} [Group G] [Fintype G]
  (hG : Fintype.card G = 1365)
  (h2G : 2 ≤ card G) (h3G : 3 < card G) :
  ¬ IsSimpleGroup G
```

The reference assumes $\mathrm{card}(G) = 1365$, which already implies the two additional cardinality bounds in the candidate. The added premises are therefore redundant, so the mathematical claim is unchanged.

Implicit binder restoration.

```
Reference:
theorem reference_statement :
  ¬ ∃ x y : ℤ, 7 * x ^ 3 + 2 = y ^ 3

Candidate:
theorem dummy :
  ¬ ∃ x : ℤ, 7 * x ^ 3 + 2 = y ^ 3
```

The candidate omits the declaration of y; Lean restores it as an implicit binder, and normalization exposes the same quantified form.