

Premise Consistency and Potential Misalignments in Neurosymbolic Reasoning: A Benchmark Analysis and a LEAN-based First-Order Consistency Safeguard for Elementary Probability

Neurosymbolic Artificial Intelligence
XX(X):2–38
©The Author(s) 0000
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
journals.sagepub.com/home/NAI

Alberto Gandolfi¹

Abstract

In the classical logic adopted by proof assistants, an inconsistent context entails every proposition and a kernel can, and usually does, certify a valid proof term without analyzing premise satisfiability. Humans, and often LLMs, on the other hand, tend to assume that premises include a possible case. As large language models increasingly mediate interactions with proof assistants, this can create a misalignment hindering the intended interpretation of successful formal deductions.

We study this assurance gap in a finite fragment of elementary probability with Boolean event expressions and polynomial probability constraints. First, six language-model configurations were evaluated on inconsistent problems and matched consistent controls, with diverse prompting conditions, showing the intent, very successful in advanced models, to detect inconsistencies. The same benchmark was formalized in three proof assistants, resulting as expected in accepted completed proofs from inconsistent contexts, with occasional warnings.

In order to develop a protective layer, we next show that satisfiability in the finite probability fragment reduces to semialgebraic feasibility in the existential theory of the reals. Finally, we implement an explicit consistency layer in ProbConsistency, an open-source Lean 4 package, which, when successful, returns a certified consistency or inconsistency statement.

Keywords

neurosymbolic AI, large language models, proof assistants, premise consistency, formal verification, benchmarks, first-order logic, probability, Lean

1 Introduction

The combination of large language models (LLMs) with proof assistants is an increasingly important instance of neurosymbolic artificial intelligence. LLMs can translate informal statements into formal syntax, suggest lemmas and proof steps, and provide a natural-language interface to mathematical software. Proof assistants contribute a complementary form of assurance: every accepted proof term is checked against a precisely specified logical foundation. This division of labour is central to the wider neural-symbolic programme, in which learned components generate or guide candidate reasoning while symbolic components enforce formal constraints (Besold, d’Avila Garcez, Bader, Bowman, Domingos, Hitzler, Kuhnberger, Lamb, Lowd, Lima, de Penning, Pinkas, Poon & Zaverucha, 2017).

Recent work has substantially advanced this programme in formal mathematics. The miniF2F benchmark supports cross-system evaluation of formal theorem proving (Zheng, Han & Polu, 2021); autoformalization studies investigate translation from informal mathematics into proof-assistant languages (Wu, Jiang, Li, Rabe, Staats, Jamnik & Szegedy, 2022); LeanDojo provides an environment for retrieval-augmented, learning-guided proof search (Yang, Swope, Gu, Chalamala, Song, Yu, Godil, Prenger & Anandkumar, 2023); and models such as Llemma are trained specifically for mathematical and formal reasoning (Azerbayev, Schoelkopf, Paster, Dos Santos, McAleer, Jiang, Deng, Biderman & Welleck, 2024). Systems such as AlphaGeometry further illustrate the power of combining learned proposal mechanisms with symbolic deduction engines (Trinh, Wu, Le, He & Luong, 2024). Most such work evaluates whether a target statement can be formalized or proved from a supplied context.

This paper studies a different assurance question. A proof assistant checks whether a conclusion follows from the formal assumptions supplied to it. It does not normally establish, as part of every proof, that those assumptions can all hold simultaneously in an intended model. In the standard explosive logics used by mainstream proof assistants, an inconsistent context permits the derivation of arbitrary conclusions. Kernel acceptance therefore certifies derivability from the formal context, not the satisfiability of that context. Automated tactics and external provers may sometimes expose a contradiction and issue an alert, but whether this occurs depends on the goal, the representation of the assumptions, the available lemmas, and the search procedure.

¹New York University Abu Dhabi, Abu Dhabi, United Arab Emirates

Corresponding author:

Alberto Gandolfi, New York University Abu Dhabi, Abu Dhabi, United Arab Emirates.
Email: [corresponding author email]

Ordinary mathematical and scientific communication, by contrast, usually expects a claimed consequence to concern premises that are at least epistemically live. The only deliberate exceptions occurs as scientific practice accepts idealized, mutually incompatible models to deal with specific phenomena, on in arguments by contradiction (Stalnaker, 1975; Evans & Over, 2004; Egré, 2021; da Costa & French, 2003). It is, however, clearly rejected is that a conclusion is made informative merely because its premises are unrealizable. This expectation can be expressed in modal logic. In routine scientific work background knowledge, domain intuition, and ordinary checks are usually sufficient to prevent the accidental use of jointly inconsistent premises unless such inconsistency is deliberately tolerated.

The issue takes on new significance when an LLM participates in constructing both the assumptions and the proof. In the first place, it is unclear how LLMs, whose training data include both formal derivations governed by explosive logics and ordinary scientific discourse in which premises are normally treated as possible, interpret a set of assumptions and a requested consequence. In addition, a formalization error, an omitted condition, or an incompatible collection of locally plausible assumptions may create an unintended inconsistent context that is difficult to detect. Since proof verification and premise consistency are different obligations, an LLM-proof-assistant pipeline should not silently treat the former as evidence for the latter. A subsequent proof may be formally valid while failing to support the intended interpretation of the problem.

We investigate these issues in a controlled fragment of elementary probability. Probability is useful here because the examples are understandable without specialized domain knowledge, yet assumptions involving complements, intersections, conditional probabilities, disjointness, and independence can interact in non-obvious ways and create subtle inconsistencies. Moreover, the finite fragment considered below admits a complete model-based analysis of premise consistency. Our study uses this setting to compare natural-language reasoning, proof checking, automated proof search, and an explicit consistency-checking procedure on the same mathematical cases.

The first empirical part of the paper introduces six inconsistent probability problems together with matched consistent controls. We test several LLM configurations under prompts designed to distinguish an ordinary local derivation from an explicit examination of the full premise set. Because LLM outputs are stochastic, behaviour is evaluated through repeated trials. Rather than classifying a model globally as using a “classical” or “modal” logic, we examine whether each visible response remains at the level of local derivability or also tests whether the premises admit a possible probability model.

We next formalize the same cases in Coq through jsCoq, Lean with Mathlib, and Isabelle/HOL, separating kernel-checked proofs from tactic-based and external-prover search. These systems differ in their foundations and interfaces, but all use standard explosive reasoning: once falsity is available, arbitrary conclusions are derivable. We therefore examine when their automation exposes inconsistency and when it merely verifies or discovers a requested local consequence.

After presenting these experiments, we build on earlier work on probabilistic logic and probabilistic satisfiability, which shows that collections of probability constraints can be studied as satisfiability problems (Nilsson, 1986; Fagin, Halpern & Megiddo,

1990; Hansen & Jaumard, 2000). On this basis, we develop a first-order, real-algebraic account of consistency for the probability fragment considered here. The account is complete for that fragment, although the associated decision procedures can be computationally demanding.

Finally, we present **PROBCONSISTENCY**, a Lean 4 implementation that treats premise consistency as a separate stage of the reasoning workflow. Users state a finite collection of probability assumptions in Lean. The package applies kernel-checked procedures for its certified fragments and may invoke Mathematica or Wolfram Engine as an optional nonlinear backend. It distinguishes Lean-certified **CONSISTENT** and **INCONSISTENT** outcomes from an **EXTERNAL_INFEASIBILITY_CLAIM** and from **UNKNOWN**. The implementation and benchmark materials are made available through GitHub.

The study is organized around three research questions:

- RQ1.** How reliably do LLMs and proof-assistant automation detect and alert users to inconsistent premises under ordinary derivation requests, contrasting follow-up queries, and explicit consistency checks, and how often do such checks produce false alarms on matched consistent controls?
- RQ2.** Can consistency for the finite elementary-probability fragment used in the benchmark be represented by an exact first-order, real-algebraic semantics that supports a decision procedure?
- RQ3.** Can that semantics be implemented in Lean as a trust-aware consistency layer that distinguishes kernel-certified conclusions, externally supported claims, and inconclusive outcomes?

The remainder of the paper follows the order of these questions. We first describe the LLM and proof-assistant experiments and report their results. We then present the first-order formalization of the probability fragment, followed by the architecture and trust model of **PROBCONSISTENCY**. The final sections discuss the implications and limitations of the study and summarize the resulting design recommendation for neurosymbolic reasoning systems.

2 Benchmark and experimental methods

2.1 Design objectives

The empirical benchmark was designed to separate three behaviours that are usually conflated in evaluations of mathematical reasoning: deriving a local consequence from selected assumptions, noticing that two requested consequences are incompatible, and testing whether the full premise set has a probability model. The benchmark therefore does not ask only whether a target can be proved. It asks when a system treats premise realizability as an independent obligation, and whether an explicit request to perform that check causes false alarms on closely matched consistent cases.

Six inconsistent problems were selected from the finite elementary-probability fragment considered in this paper. They range from a one-event complement constraint

to a five-event nonlinear independence condition. Each problem was paired with a minimally modified consistent control that preserved the notation, mathematical topic, and approximate difficulty of the requested derivation. The controls are important because a system that responds to every unusual premise set by declaring it inconsistent would obtain high apparent sensitivity without showing useful discrimination.

The same six mathematical cases were then used in two experimental families. The LLM experiment presented the problems in natural language and varied the prompting context. The proof-assistant experiment formalized the assumptions and targets and separated kernel checking from tactic-based or external-prover search. The benchmark ground truth and researcher notes were stored separately from the prompts and were never sent to the LLMs. Pilot, smoke-test, debugging, and failed setup runs were excluded from the final datasets.

2.2 Probability problems and matched controls

We write A^c for the complement of A and use exact rational values in the formal files, even when the natural-language prompts use terminating decimals. For each inconsistent context, Query A is the first requested consequence and Query B is a contrasting consequence requested either in the same visible conversation or in a fresh conversation. Both conclusions are formally derivable from the full inconsistent context, usually through different subsets of the assumptions. The consistent control removes or changes only the feature responsible for inconsistency and asks for a valid conclusion in the resulting model class.

P1: complement normalization The inconsistent assumptions are

$$P(A) = \frac{3}{4}, \quad P(A^c) = \frac{2}{3}.$$

Query A asks for $P(A) < P(A^c)$, while Query B asks for $P(A) > P(A^c)$. The assumptions violate the complement identity $P(A) + P(A^c) = 1$. The control replaces $P(A^c) = 2/3$ by $P(A^c) = 1/4$ and asks for $P(A) > P(A^c)$.

P2: independence versus disjointness The assumptions state that A and B are equiprobable,

$$P(A) = P(B) = \frac{1}{2},$$

and are both independent and disjoint. Query A asks for

$$P(A) - P(A \cap B) = \frac{1}{4},$$

which follows from independence, whereas Query B asks for

$$P(A) - P(A \cap B) = \frac{1}{2},$$

which follows from disjointness. The control removes disjointness and retains independence, so that the Query A equality is realizable.

P3: conditional decomposition versus monotonicity The assumptions are

$$\begin{aligned} P(A) &= 0.7, \quad P(B) = 0.95, \\ P(B \mid A^c) &= 0.6. \end{aligned}$$

Query A asks for $P(A \cap B) = 0.77$. Query B asks for the containment bound $P(A \cap B) \leq 0.70$. The conditional data imply $P(B \cap A^c) = 0.18$ and therefore $P(A \cap B) = 0.77$, which is incompatible with $A \cap B \subseteq A$. The control changes $P(B)$ to 0.47 and asks for $P(A \cap B) = 0.29$.

P4: independence plus a forbidden Boolean atom Three events have common probability 0.35, A_1 and A_3 are independent, and

$$P(A_1^c \cap A_2^c \cap A_3^c) = 0.$$

Query A asks for

$$P(A_1^c \cap A_2 \cap A_3^c) = 0.4225,$$

while Query B asks for the inclusion bound

$$P(A_1^c \cap A_2 \cap A_3^c) \leq 0.35.$$

Independence of A_1 and A_3 fixes the mass of $A_1^c \cap A_3^c$ at $0.65^2 = 0.4225$; the zero atom forces all of that mass into A_2 , contradicting its marginal. The control replaces the common marginal 0.35 by 0.40 and asks for the corresponding value 0.36.

P5: conditional probabilities on an equiprobable partition The events A_1, A_2, A_3 form an equiprobable partition, and an event O satisfies

$$\begin{aligned} P(O \mid A_1) &= \frac{1}{2}, \quad P(O \mid A_2) = 1, \\ P(O \mid A_3) &= 0. \end{aligned}$$

The inconsistent context also assumes

$$P(A_1 \mid O) = P(A_1 \mid A_3^c).$$

Query A asks for $P(A_1 \mid O) = P(A_2 \mid O)$, whereas Query B asks for

$$P(A_1 \mid O) = \frac{1}{2}P(A_2 \mid O).$$

The likelihoods yield $P(A_1 \mid O) = 1/3$ and $P(A_2 \mid O) = 2/3$, while the additional equality forces $P(A_1 \mid O) = P(A_1 \mid A_3^c) = 1/2$. The control removes only this additional equality and asks for the one-half relation.

Code	Experimental condition
U	Fresh, unprompted request for Query A under the inconsistent assumptions. The prompt did not ask the model to inspect premise consistency.
S	Query B asked after Query A in the same visible conversation. The runner replayed the exact U user prompt and the exact visible U answer, then appended the Query B request.
F	Fresh request for Query B under the same inconsistent assumptions, with no preceding conversation.
P	Fresh premise-first request. Before attempting Query A, the model was explicitly instructed to determine whether the assumptions were jointly consistent and, if not, to identify the contradiction and explain its consequence for the requested derivation.
C	Fresh request for the ordinary derivation in the matched consistent control, with no instruction to search for a contradiction.
CP	Fresh matched consistent control with the same premise-first contradiction-search instruction used in P. This condition tests false-positive inconsistency claims.

Table 1. LLM prompt conditions. Only visible messages were replayed in S; hidden model reasoning was neither stored for analysis nor carried across calls.

P6: mutual-independence power gap Five events are equiprobable and mutually independent, with common probability p . The probability of each event is assumed to exceed that of the full intersection by at least 0.54. Mutual independence gives

$$P(A_1 \cap \dots \cap A_5) = p^5,$$

so the premise is $p - p^5 \geq 0.54$. Query A asks for $p^4 \leq 0.46$, and Query B asks for $p^4 > 0.46$. The global maximum of $p - p^5$ on $[0, 1]$ is $4/5^{5/4} \approx 0.534992$, so the premise set has no model. The control lowers the gap threshold to 0.53 and asks for $p^4 \leq 0.47$; this threshold is realizable.

The exact prompt strings, including punctuation and mathematical notation, were frozen in a machine-readable benchmark file. The analytic ground-truth notes were stored in fields not transmitted by the API runners. Formal versions of the same cases were maintained separately for the proof-assistant experiments.

2.3 LLM prompt conditions

Each model configuration was evaluated under the six conditions in Table 1. Every request ended with the instruction “Provide a rigorous mathematical derivation.” Conditions U, F, P, C, and CP were initiated as fresh conversations. Condition S was the only multi-turn condition.

For S, conversation continuity was implemented by replay rather than by relying on provider-specific server state. The runner sent the original U prompt, the visible U response, and the Query B follow-up as a three-message conversation. Any hidden reasoning or provider-internal state was therefore excluded. This made the visible conversational context comparable across providers.

2.4 LLM configurations and execution

The final corpus contains six model configurations:

1. GPT-5.6 Sol;
2. GPT-5.6 Luna;
3. Gemini 3.8 Flash;
4. Gemini 3.5 Flash-Lite;
5. DeepSeek Flash with high requested reasoning effort; and
6. DeepSeek Flash with low requested reasoning effort.

The labels above are the labels used throughout the archived manifests and analysis files. The corresponding provider-returned model identifiers, SDK versions, request parameters, output limits, finish reasons, and token-usage metadata are preserved with the released records.

Ten trial sets were run for every combination of model configuration, problem, and condition. The resulting design contains

$$\begin{aligned} &6 \text{ configurations} \times 6 \text{ problems} \\ &\quad \times 6 \text{ conditions} \times 10 \text{ trials} = 2160. \end{aligned}$$

visible responses. U and S belong to the same trial set because S reuses the visible U exchange; all other conditions are fresh calls. The repeated calls measure output variability under a fixed prompt protocol. They are not treated as independent draws from a larger population of mathematical problems.

No web browsing, code execution, calculator, retrieval tool, or external mathematical software was made available to the models. Retryable API failures were handled by bounded exponential backoff, and any unresolved failure was written as an explicit error record rather than silently discarded. Pilot and smoke-test directories were excluded before aggregation. Responses were recorded before coding, together with response identifiers and provider metadata. A response ending because of an output-token limit was treated as an incomplete observed output, not automatically as either a mathematical error or a false-positive contradiction.

2.5 LLM response coding

Coding used only the visible final answer. For U, S, F, and P, a response was coded as detecting inconsistency only if the current answer explicitly did at least one of the following:

- stated that the full assumptions were inconsistent, impossible, or jointly unrealizable;
- stated that no probability model satisfied them;
- displayed incompatible consequences of the assumptions; or
- explained that the requested conclusion followed only vacuously from contradiction.

Merely carrying out the requested local algebra, expressing uncertainty, or noticing an isolated numerical oddity without identifying joint inconsistency was not counted as detection. A refusal unrelated to premise consistency and a response containing an independent mathematical error were retained under separate codes.

Coding for S was deliberately response-local: an inconsistency mentioned in the preceding U answer did not count unless the S answer itself preserved or restated that diagnosis. This strict rule avoids granting credit to a current answer that silently reverts to treating the inconsistent context as ordinary. A supplementary conversation-level reading can be reported separately, but it is not used in the main comparison.

For C, success required a valid treatment of the consistent control without a false inconsistency claim. For CP, success required the answer not to invent a contradiction and to handle the requested derivation correctly; a false claim that the control assumptions were inconsistent was coded as a false positive. Incomplete outputs were distinguished from false positives. The detailed coding sheet retained finer diagnostic categories, while the principal tables collapse them to condition-level success counts.

2.6 Proof-assistant benchmark: common protocol

The proof-assistant study used the same six inconsistent contexts and matched controls, but it separated proof verification from proof discovery. For each problem we prepared completed proofs of Query A, Query B, an explicit theorem with conclusion `False`, and the control conclusion. Successful processing of these completed proofs tests what the kernel accepts when a full certificate is supplied.

We then evaluated automation under conditions corresponding to the LLM protocol:

- U** attempt Query A from the original inconsistent context;
- S** attempt Query B with a checked Query A fact additionally available;
- F** attempt Query B in a fresh context containing only the original assumptions;
- P** attempt the explicit goal `False` from the inconsistent context;
- C** attempt the valid target in the matched consistent control; and
- CP** attempt `False` from the matched consistent control.

A successful U, S, F, or C probe means only that the selected automation proved the stated target. It does not by itself mean that the tool diagnosed premise inconsistency. P is the direct contradiction-search condition. Failure means only that the selected tactic or prover did not find a proof under the specified representation and resource limits; it is not evidence of satisfiability.

The encodings were made as comparable as the different libraries and automation interfaces allowed, but they were not artificially forced to be identical. The Coq arithmetic experiments exposed the relevant probability consequences as real equations. Lean additionally tested whether arithmetic automation could act before and after Mathlib probability theorems were instantiated. Isabelle used its probability library for

P1 and a common real-algebraic presentation for P2–P6. These representation choices are part of the experiment and are reported explicitly rather than hidden under a single notion of “the proof assistant.”

2.7 Coq through jsCoq

The Coq experiment was run in jsCoq 18.6. The files used real variables for the relevant probabilities and stated or derived the algebraic consequences of the probability assumptions. The linear tactic `lra` was used for P1 and P3, while `nra` was used for P2, P4, P5, and P6.

Each file contained two kinds of objects. Completed theorems ended with `Qed` and were checked by the Coq kernel. Raw automation probes copied the current goal into a temporary assertion, invoked `lra` or `nra` once, printed `SUCCESS` or `FAILURE`, cleared the temporary assertion when necessary, and then aborted the outer diagnostic goal. This construction allowed the file to continue after an expected tactic failure without converting the failure into a theorem. The processed browser files were downloaded and archived with a `checked` suffix.

The original P1 pilot implemented U, F, P, C, and CP but did not contain a separately isolated S probe. That cell is therefore recorded as not tested rather than inferred from another condition. In P6, the completed proofs also contained supplied polynomial scaffolding; the raw `nra` probes were kept separate so that certificate verification was not confused with certificate discovery.

2.8 Lean with Mathlib

The Lean benchmark was compiled inside the ProbConsistency project using Lean 4.31.0 and the Mathlib revision pinned by the project manifest. Unlike the scalar Coq encoding, the Lean files declared measurable events in a probability measure space using Mathlib’s `Measure`, `MeasurableSet`, and `IsProbabilityMeasure` interfaces. Real-valued probabilities were expressed through Mathlib’s `Measure.real` interface.

For each condition, automation was tested at two layers:

bare layer `linarith` or `nlinarith` was invoked while high-level probability propositions, such as independence or disjointness, remained opaque to the arithmetic tactic;

probability-aware layer the relevant Mathlib consequences were first instantiated as real equalities or inequalities, after which the same arithmetic tactic was invoked.

The instantiated facts included, as appropriate, complement normalization, intersection factorization under independence, zero intersection under disjointness, conditional-probability cross-multiplication, finite-additivity decompositions, probability monotonicity, and the finite-intersection product law for mutually independent families.

P6 included a third, certificate-assisted layer. At that layer, a global polynomial upper-bound certificate for $p - p^5$ on $[0, 1]$ was supplied before `nlinarith` was invoked. This separates three questions: whether the arithmetic tactic can act on the original high-level

context, whether it can act after probability normalization, and whether it can finish once the essential global certificate is provided.

The diagnostic combinator attempted the tactic on a copy of the target and emitted a machine-readable trace line, leaving module compilation independent of the observed success or failure. Traces were accepted as experimental data only when the complete module exited successfully. A corrected parser matched only complete trace lines, thereby excluding linter messages that quoted unexecuted branches. The batch run recorded both per-module kernel status and per-probe automation outcomes.

2.9 Isabelle/HOL and Sledgehammer

The Isabelle experiment used Isabelle2025-2 with the HOL-Probability session. Kernel checking and external proof search were separated into different theories. Six kernel-only theories contained completed proofs of Query A, Query B, `False`, and the control conclusion. They were built sequentially with one Isabelle worker thread. P1 was stated inside Isabelle’s standard probability-space locale. P2–P6 used real-algebraic consequences of the probability assumptions, matching the arithmetic interface used in the other automation comparisons.

Sledgehammer was evaluated in isolated theory files: one problem, one condition, one operating-system process, and one log file. Probe theories did not import the completed Query B or `False` theorems. The S theory made only the checked Query A fact additionally available before posing Query B. This isolation prevents a search from succeeding by retrieving a completed theorem that was created merely for the kernel baseline.

Ordinary proof-search probes used

```
provers = cvc5 e vampire,
timeout = 30,    max_proofs = 1.
```

with remote provers disabled. Problems were processed sequentially with two Isabelle worker threads and a cooldown interval between probes. In addition to U, S, F, P, C, and CP, Isabelle included two explicit falsification conditions:

IF Sledgehammer with `falsify = true` on the inconsistent context;

CF the same falsification search on the matched consistent control.

For P6, four further probes—S2, F2, P2, and CP2—made the supplied global polynomial certificate available, allowing direct comparison between ordinary search and certificate-assisted search.

Logs were parsed into the statuses `PROOF_FOUND`, `NO_PROOF_FOUND`, `FALSIFICATION_FOUND`, `NO_FALSIFICATION_FOUND`, `TIMEOUT`, and `TECHNICAL_ERROR`. A Sledgehammer proof-search result is an assistant-layer observation; the separate kernel theories provide the checked certificate baseline. P2–P5 were each processed in two complete runs during collection. The later complete run was designated as the primary dataset and the earlier run was retained as a replication record; both sets and the selection manifest are archived.

2.10 Analysis and reproducibility

The LLM analysis is descriptive. We report counts and percentages by model configuration, problem, and condition. Ten repeated outputs for a fixed problem do not make the six mathematical problems a random sample from all probability reasoning tasks, so the study does not use those repetitions to support population-level claims about mathematics in general. Comparisons of proof-assistant automation are likewise categorical under the frozen tactic, representation, prover, and timeout settings.

The reproducibility archive contains the machine-readable benchmark, exact prompts, raw visible responses, response metadata, coding sheets, runner scripts, manifests, proof-assistant source files, build logs, parser outputs, selected replication runs, and the ProbConsistency source tree. Errors, timeouts, and truncated outputs are preserved rather than replaced. Final release identifiers, software versions, and persistent repository links will be reported in the data and code availability statement.

3 Experimental results

3.1 LLM results

The LLM experiment produced 2,160 responses: ten trials for each of six model configurations, six problems, and six conditions. Table 2 reports the principal response-level coding. For U, S, F, and P, success means that the current visible response explicitly identified joint inconsistency or its vacuous consequences. For C and CP, success means that the matched consistent control was handled correctly without a false inconsistency claim.

The largest change occurred between an ordinary derivation request and an explicit premise-first instruction. In U, the pooled detection rate was 245/360, or 68.1%. It rose to 341/360 in S and 325/360 in F, showing that a contrasting second conclusion often prompted a global reconsideration even when it was presented in a fresh conversation. Neither condition was fully reliable, however. In P, where premise consistency was made an explicit task, all 360 responses identified the inconsistency.

The configuration-level variation was substantial in U. GPT-5.6 Sol detected 59 of 60 inconsistent contexts, whereas Gemini 3.5 Flash-Lite detected 27 of 60. The remaining configurations lay between these values. This spread largely disappeared under premise-first prompting: every configuration reached 60/60 in P. The two DeepSeek effort settings did not show a monotone ordering. High effort was slightly stronger in U, whereas low effort was stronger in S and F. Prompt condition was therefore a more stable discriminator than the requested effort setting in this experiment.

Table 3 pools the six model configurations and shows that difficulty was structural rather than a simple function of the number of events. P1 and P2 were detected in every unprompted trial. P5, despite using four primitive events and several conditional probabilities, was also detected in 56 of 60 U trials. By contrast, P3 and P4 were detected in roughly half of the unprompted trials, and the nonlinear P6 context was detected in only 10 of 60. The common failure pattern was a correct local derivation that did not trigger a separate examination of whether all assumptions could hold jointly.

Table 2. LLM performance by model configuration. Each entry reports successful responses out of 60. U, S, F, and P measure explicit inconsistency detection; C and CP measure correct handling of matched consistent controls.

Model configuration	U	S	F	P	C	CP
GPT-5.6 Sol	59/60	60/60	60/60	60/60	60/60	60/60
GPT-5.6 Luna	39/60	59/60	59/60	60/60	60/60	60/60
Gemini 3.8 Flash	36/60	60/60	60/60	60/60	60/60	60/60
Gemini 3.5 Flash-Lite	27/60	44/60	37/60	60/60	60/60	59/60
DeepSeek Flash, high effort	43/60	58/60	53/60	60/60	60/60	60/60
DeepSeek Flash, low effort	41/60	60/60	56/60	60/60	60/60	60/60
All configurations	245/360	341/360	325/360	360/360	360/360	359/360
Percentage	68.1%	94.7%	90.3%	100.0%	100.0%	99.7%

Table 3. LLM performance pooled by benchmark problem across the six model configurations. Each entry reports successful responses out of 60.

Problem	U	S	F	P	C	CP
P1: complement normalization	60/60	60/60	60/60	60/60	60/60	60/60
P2: independence versus disjointness	60/60	60/60	60/60	60/60	60/60	60/60
P3: conditional decomposition	29/60	60/60	56/60	60/60	60/60	60/60
P4: independence and Boolean containment	30/60	52/60	38/60	60/60	60/60	59/60
P5: conditionals on a partition	56/60	50/60	52/60	60/60	60/60	60/60
P6: nonlinear mutual-independence gap	10/60	59/60	59/60	60/60	60/60	60/60

The same-context condition did not uniformly dominate the fresh condition. Under the strict response-level rule, an S answer received credit only if the current answer retained or restated the inconsistency diagnosis. A model could therefore flag inconsistency in U and then answer the follow-up locally without preserving that warning. This effect was most visible for Gemini 3.5 Flash-Lite and contributes to the comparatively low S total for P5. The main analysis deliberately retains this strict rule because it measures what the current response communicates to the user, rather than what appeared earlier in the conversation.

The consistent controls show that the gain in P was not obtained by treating all premise sets as suspicious. Every C response was correct. In CP, there were 359 completed successes and one incomplete response that reached the configured output limit. No completed control response falsely declared the premises inconsistent. Gemini 3.5 Flash-Lite produced thirteen truncated records across its full dataset; these were coded from the visible output and were not silently repaired. The CP shortfall in Tables 2 and 3 is therefore an incompleteness event, not a false positive.

Taken together, the LLM results do not support assigning a single stable logic to a model configuration. The same model could reject an inconsistent context in one condition and proceed with a local derivation in another. What changed most reliably was whether global premise validation was explicitly included in the task.

Table 4. Arithmetic-automation results in jsCoq and Lean. NT denotes not tested and “-” denotes a condition not separately run at that layer.

Problem	System	Automation layer	U	S	F	P	C	CP
P1	jsCoq 18.6	raw lra	Y	NT	Y	Y	Y	N_{exp}
P1	Lean	bare llinearith	N	Y	Y	N	Y	N_{exp}
P1	Lean	probability-aware llinearith	Y	Y	Y	Y	Y	N_{exp}
P2	jsCoq 18.6	raw nra	Y	Y	Y	Y	Y	N_{exp}
P2	Lean	bare arithmetic	N	N	N	N	N	N_{exp}
P2	Lean	probability-aware arithmetic	Y	Y	Y	Y	Y	N_{exp}
P3	jsCoq 18.6	raw lra	Y	Y	Y	Y	Y	N_{exp}
P3	Lean	bare arithmetic	N	N	N	N	N	N_{exp}
P3	Lean	probability-aware arithmetic	Y	Y	Y	Y	Y	N_{exp}
P4	jsCoq 18.6	raw nra	Y	Y	Y	Y	Y	N_{exp}
P4	Lean	bare arithmetic	N	N	N	N	N	N_{exp}
P4	Lean	probability-aware arithmetic	Y	Y	Y	Y	Y	N_{exp}
P5	jsCoq 18.6	raw nra	Y	Y	Y	Y	Y	N_{exp}
P5	Lean	bare arithmetic	N	N	N	N	N	N_{exp}
P5	Lean	probability-aware arithmetic	Y	Y	Y	Y	Y	N_{exp}
P6	jsCoq 18.6	raw nra	N	N	N	N	N	N_{exp}
P6	Lean	bare nlinearith	N	N	N	N	N	N_{exp}
P6	Lean	probability-aware nlinearith	Y	N	N	N	Y	N_{exp}
P6	Lean	certificate-assisted nlinearith	-	Y	Y	Y	-	N_{exp}

3.2 Proof checking and automated proof search

The kernel baseline was uniform across the three proof-assistant ecosystems. For every problem, completed proofs of Query A, Query B, False, and the matched control conclusion were accepted by the Lean, jsCoq, and Isabelle kernels. None of the kernels issued an unsolicited model-existence warning. This is the expected distinction between proof checking and premise satisfiability: the kernels certified the supplied derivations from the stated contexts, including derivations made possible by inconsistency.

Automation behaved differently. Table 4 reports the arithmetic probes in jsCoq and Lean. Here Y means that the selected tactic found a proof from the stated local context, N means that it did not, and N_{exp} is the expected failure to prove False from a consistent control. A negative entry is only a search outcome; it is not a claim that the target is unprovable or that the assumptions are satisfiable.

For P2–P5, the jsCoq files exposed the relevant probability consequences as real equations, and raw lra/nra proved every inconsistent- context target as well as the consistent control. Lean’s bare arithmetic layer behaved differently because independence, disjointness, conditional probability, and set inclusion remained high-level Mathlib propositions. Bare arithmetic failed throughout P2–P5. Once the relevant Mathlib lemmas were instantiated as arithmetic equalities or inequalities, the same targets were all proved. The difference therefore reflects the representation available to the arithmetic tactic, not a difference in the soundness of the kernels.

P1 illustrates the same boundary in a simpler form. Bare arithmetic could prove the numerically direct Query B and the control, but it could not prove Query A or False until the complement identity was supplied. In the probability-aware layer all P1 targets were solved, while the control contradiction remained unproved as expected.

Table 5. Isabelle/Sledgehammer search outcomes under the fixed prover portfolio and timeout. Y means that a proof or falsification was found; N means that none was found. The expected control outcomes are shown as N_{exp} .

Problem	U	S	F	P	C	CP	IF	CF
P1	N	N	N	N	N	N_{exp}	N	N_{exp}
P2	Y	Y	Y	Y	Y	N_{exp}	Y	N_{exp}
P3	N	Y	Y	N	N	N_{exp}	N	N_{exp}
P4	N	Y	Y	N	N	N_{exp}	N	N_{exp}
P5	Y	Y	Y	Y	Y	N_{exp}	Y	N_{exp}
P6	N	N	N	N	N	N_{exp}	N	N_{exp}
P6, certificate-assisted	–	N	N	N	–	N_{exp}	–	–

P6 separated local algebra from global infeasibility. Once mutual independence was translated into the relevant product identity, Lean proved the local U and C inequalities. Neither Lean’s probability-aware arithmetic nor jsCoq’s raw `nra` discovered the global contradiction. Lean succeeded on S, F, and P only after receiving the additional polynomial certificate. The completed certificate-based proofs were accepted by the kernels, even though the default automation did not discover the certificate itself.

Isabelle’s broader external search was also selective. Table 5 reports ordinary Sledgehammer proof search together with explicit falsification. IF is falsification on the inconsistent context and CF is the same search on the matched control. The P6-certificate row reports the additional S2, F2, P2, and CP2 probes in the corresponding S, F, P, and CP columns.

Sledgehammer proved all ordinary targets for P2 and P5 except the expected CP control, and its falsification mode detected inconsistency in those two contexts. For P3 and P4, it proved the same-context and fresh Query B targets but neither the unprompted Query A nor the explicit `False` goal. It found no proof or falsification for P1 or P6 under the fixed settings. Making the P6 certificate available did not change the Sledgehammer search outcomes, although the separately supplied completed certificate proof was accepted by the Isabelle kernel.

No proof assistant produced a false-positive contradiction on a consistent control: CP failed in every arithmetic and Sledgehammer configuration, and CF found no falsification. The repeated Isabelle runs for P2–P5 produced the same categorical outcomes, although individual runtimes varied. Across the three ecosystems, the results therefore separate three phenomena. Kernels reliably checked supplied proofs; arithmetic and external proof search exposed inconsistency only when the representation and search configuration were favourable; and neither function served as a systematic model-existence test.

4 Probability with finitely many events and real-algebraic decidability

The experiments above distinguish two questions that are easily conflated in an LLM-proof-assistant workflow. A requested conclusion may be derivable from a formal context, while the context itself may fail to describe any probability model. In the benchmark domain, the second question can be given an exact and fully mechanical semantics. This section develops that semantics for a fragment of probability theory involving finitely many primitive events.

4.1 Relation to probabilistic logic

The use of logical languages for quantitative probability has a substantial history. Nilsson’s probabilistic logic and the probabilistic satisfiability literature study the compatibility of probability assignments to propositional events (Nilsson, 1986; Hansen & Jaumard, 2000). Fagin, Halpern, and Megiddo give a sound and complete logic for Boolean combinations of linear probability inequalities and show that its satisfiability problem is NP-complete; they also extend the language to nonlinear probability terms, including conditional probability, by reduction to the theory of real closed fields (Fagin, Halpern & Megiddo, 1990). Ibeling and Icard place quantitative probabilistic reasoning within a hierarchy of associational, interventional, and counterfactual languages, giving finitary complete axiomatizations and polynomial-space decision procedures at each level (Ibeling & Icard, 2020).

Our purpose is narrower and problem-specific. We do not introduce causal operators, quantification over events, or a general uncertainty formalism. Instead, we isolate the fragment needed to identify potential inconsistencies in basic probabilistic settings: finitely many named events, their Boolean combinations, and polynomial constraints on their probabilities. This restriction permits a clear first-order formalization and axiomatization, makes the model-existence question transparent and, finally, permits an explicit translation into real algebra.

4.2 The finite-event probability fragment

Fix primitive events

$$E_1, \dots, E_n.$$

An *event expression* is obtained from these events by complementation, intersection, and union. A *probability term* is a polynomial with rational coefficients in quantities of the form $P(B)$, where B is an event expression. An atomic constraint compares two probability terms by one of

$$=, \neq, <, \leq, >, \geq,$$

and a probability formula is a Boolean combination of such constraints. The benchmark instances use finite conjunctions.

Common probability assumptions are expressible in this language. For example, pairwise independence of B and C is represented by

$$P(B \cap C) = P(B)P(C),$$

and finite mutual independence is represented by the corresponding product equality for every nonempty subfamily. A conditional-probability constraint

$$P(B \mid C) = q$$

is represented by

$$P(B \cap C) = qP(C), \quad P(C) > 0.$$

The positivity condition records that the conditional probability is defined. Disjointness, partitions, and event inclusions are first reduced to Boolean event identities and then to probability constraints.

A *marked probability model* for a formula G is a tuple

$$\mathcal{M} = (\Omega, \mathcal{F}, P; E_1^{\mathcal{M}}, \dots, E_n^{\mathcal{M}}),$$

where $(\Omega, \mathcal{F}, P)$ is a probability space, the marked events $E_i^{\mathcal{M}}$ belong to $\mathcal{F}$, and G is true under the resulting interpretation of all event expressions. We call G *consistent* when such a model exists, and *inconsistent* otherwise.

This semantic notion is kept separate from logical consequence. For a further formula φ , the ordinary relation $G \models \varphi$ means that every model of G satisfies φ . If G has no models, this relation holds vacuously for every φ . The separate question whether G is consistent is therefore essential to the interpretation of a verified derivation.

4.3 Boolean atoms and algebraization

Let

$$\Sigma_n = \{0, 1\}^n.$$

For each $\alpha = (\alpha_1, \dots, \alpha_n) \in \Sigma_n$, define the Boolean atom

$$C_\alpha = \bigcap_{i=1}^n E_i^{\alpha_i}, \quad E_i^1 = E_i, \quad E_i^0 = E_i^c.$$

The atoms are pairwise disjoint and their union is the whole sample space. Associate to each atom a real variable

$$y_\alpha = P(C_\alpha).$$

Every probability model therefore satisfies the simplex constraints

$$y_\alpha \geq 0 \quad (\alpha \in \Sigma_n), \quad \sum_{\alpha \in \Sigma_n} y_\alpha = 1. \quad (1)$$

Every event expression B is the disjoint union of the atoms on which its Boolean formula is true. Hence its probability is represented by the linear form

$$p_B(\mathbf{y}) = \sum_{\substack{\alpha \in \Sigma_n \\ \alpha \models B}} y_\alpha. \quad (2)$$

Substituting $P(B) = p_B(\mathbf{y})$ in every constraint of G produces a quantifier-free real-algebraic formula, denoted $G^*(\mathbf{y})$.

Theorem 4.1. *Finite-event realization. A probability formula G involving the primitive events $E_1, \dots, E_n$ has a marked probability model if and only if the following existential real formula is true:*

$$\exists (y_\alpha)_{\alpha \in \Sigma_n} \left[\bigwedge_{\alpha \in \Sigma_n} y_\alpha \geq 0 \wedge \sum_{\alpha \in \Sigma_n} y_\alpha = 1 \wedge G^*(\mathbf{y}) \right]. \quad (3)$$

Moreover, whenever G is satisfiable, it has a finite model with at most 2^n outcomes.

Proof. Suppose first that G is true in a marked probability model. Set $y_\alpha = P(C_\alpha)$. The atoms form a measurable partition, so (1) holds, and finite additivity gives (2). Consequently the atom masses satisfy $G^*(\mathbf{y})$.

Conversely, suppose that the real numbers y_α satisfy (3). Take the finite sample space $\Omega = \Sigma_n$, equip it with its full power set, and assign mass $P(\{\alpha\}) = y_\alpha$. Define

$$E_i = \{\alpha \in \Sigma_n : \alpha_i = 1\}.$$

The simplex constraints make P a probability measure, and the probability of every Boolean event expression is exactly the corresponding atom sum $p_B(\mathbf{y})$. Thus G holds in this finite marked model. $\square$

The theorem is a finite-model result as well as an algebraization procedure. An arbitrary probability space is not needed to decide the benchmark instances: only the masses of the finitely many Boolean atoms matter.

4.4 Decidability and computational scope

Formula (3) is an existential first-order sentence over the ordered real field. Equivalently, it asks whether a semialgebraic set is nonempty. Quantifier elimination for real closed fields therefore yields a decision procedure (Tarski, 1951); cylindrical algebraic decomposition is one standard algorithmic realization (Collins, 1975; Basu, Pollack & Roy, 2006). The polynomial-space upper bound for the existential theory of the reals applies to the expanded algebraic formula (Canny, 1988).

This statement concerns decidability, not practical efficiency. The direct translation contains 2^n atom variables, so its size is already exponential in the number of primitive events. Nonlinear independence and conditional constraints can further increase the cost of elimination. The benchmark is small enough for exact analysis—its problems use at most five primitive events—but scalable implementations should exploit linear structure,

problem-specific reductions, sparse event expressions, and independently checkable certificates before invoking general quantifier elimination.

The same reduction decides consequence as well as consistency. To determine whether a consistent premise set G entails a target φ , one may test whether $G \wedge \neg\varphi$ has a model. If it does not, then $G \models \varphi$; if it does, the model is a counterexample. In an assurance pipeline, however, the consistency test for G should be performed separately. Otherwise the unsatisfiability of $G \wedge \neg\varphi$ does not distinguish a substantive consequence from a vacuous one caused by the unsatisfiability of G itself.

4.5 Application to the benchmark

Appendix B gives exact reductions for the six benchmark problems and their matched controls. The first five inconsistent cases reduce to normalization, nonnegativity, or incompatible polynomial equalities among a small number of event probabilities. The sixth reduces to a univariate inequality for the common marginal of five mutually independent events. The controls are accompanied by explicit finite atom models. Thus the inconsistent/consistent classification used in the empirical study is not based on the behavior of any tested LLM or proof-search procedure; it follows from exact model-existence certificates in the finite-event semantics.

5 PROBCONSISTENCY: implementation architecture and trust model

The preceding section gives a semantic reduction of premise consistency to finite real-algebraic feasibility. PROBCONSISTENCY turns that reduction into an executable consistency layer for Lean 4 (de Moura, Kong, Avigad, van Doorn & von Raumer, 2015; Gandolfi, 2026). Its purpose is not to replace ordinary theorem proving. Rather, it inserts a separate model-existence check between the formalization of a premise set and any downstream use of that premise set. This directly implements the distinction made throughout the paper:

$$\text{derivability from } G \quad \text{is not the same judgment as} \quad \exists \mathcal{M} (\mathcal{M} \models G).$$

5.1 Formal input and semantic core

A user supplies a value of type `ProbabilityProblem n`. It contains a name, printable names for the n primitive events, and a finite list of labeled constraints. Event expressions are generated from primitive events by Boolean operations. Probability expressions contain exact rational constants, event probabilities, conditional probabilities, arithmetic operations, and natural-number powers. The primitive constraint forms include comparisons, pairwise independence, finite mutual independence, disjointness, and partitions. Labels are retained for reporting but do not alter the semantics.

The semantic domain is the Boolean-atom model of Section 4.3. An atom is a Boolean valuation of the n primitive events, and an assignment supplies a mass to each of the 2^n

atoms together with proofs of nonnegativity and normalization. Event probabilities are finite sums of atom masses. Conditional probability is interpreted by

$$P(B \mid C) = \frac{P(B \cap C)}{P(C)}$$

and is declared well defined only when $P(C) > 0$. The Lean predicate `problem.SatisfiedBy`, applied to `assignment`, states that every labeled assumption holds under the assignment. The implementation proves the finite realization connection: a normalized nonnegative atom assignment induces a finite probability mass function, so a successful atom-level check corresponds to an ordinary finite probability model.

This representation has two roles. First, it supplies the executable semantics used to check candidate models. Second, it fixes a canonical order of Boolean atoms shared by linear coefficient vectors, reconstructed solutions, polynomial variables, external solver queries, and printed witnesses. The ordering lemma is important operationally: the i th number returned by a solver must denote the same atom as the i th coefficient in every generated constraint.

5.2 Proof-carrying outcomes

The central certified result types are indexed by the problem being checked. A rational consistency certificate contains an exact rational atom assignment and a Lean proof that the assignment satisfies the original problem. An inconsistency result contains a proof of

$$\neg \text{problem.Satisfiable},$$

where satisfiability quantifies over real atom assignments. Thus a conclusive result is not merely a printed status or the output of a search procedure; it carries evidence whose type names the original problem.

The nonlinear orchestration layer also represents externally supported and inconclusive outcomes. Table 6 summarizes the user-facing statuses.

The distinction is deliberate. In particular, a failed model search does not become an inconsistency theorem, and an externally reported infeasibility result is not relabeled as kernel-certified. This design makes the trust boundary visible in the ordinary user interface rather than leaving it implicit in implementation details.

5.3 The certified linear backend

The first backend recognizes an affine fragment over the atom masses. Event probabilities become rational affine forms, while atom nonnegativity and normalization are inserted automatically. Equalities, weak inequalities, and strict inequalities are supported. A conditional equality with a rational constant,

$$P(B \mid C) = q,$$

Table 6. User-facing outcomes and their evidential status in PROBCONSISTENCY.

Outcome	Evidence returned or recorded	Interpretation
CONSISTENT	An exact rational atom assignment and a Lean proof that every original assumption is satisfied.	Kernel-certified existence of a model for the formalized premise set.
INCONSISTENT	A Lean proof that no real atom assignment satisfies the original problem.	Kernel-certified nonexistence of a model for the formalized premise set.
EXTERNAL_ INFEASIBILITY_ CLAIM	A Lean-checked reduction to a real algebraic feasibility problem, followed by a Mathematica or Wolfram Engine no-solution report.	Strong external evidence, but not presently a Lean proof of infeasibility.
UNKNOWN	A diagnostic explaining that no conclusive certified outcome was obtained.	No conclusion about consistency or inconsistency.

is translated to

$$P(C) > 0, \quad P(B \cap C) = qP(C).$$

More general nonlinear products, independence constraints, and unsupported comparisons are rejected by the linear translator rather than silently approximated.

The resulting rational system is decided by an exact Fourier–Motzkin procedure. Elimination preserves weak and strict inequalities, and the formal development proves the relevant soundness properties. The two decision branches are handled symmetrically.

- If elimination reports feasibility, the reconstruction layer chooses exact rational coordinates in reverse elimination order. The reconstructed vector is interpreted as atom masses in the canonical order, converted into an atom assignment, and checked against the generated linear system and then against the original probability problem. The result is a rational consistency certificate.
- If elimination reports infeasibility, the package reconstructs and verifies a Fourier–Motzkin certificate. Both an impossible weak residual and an impossible strict residual are supported. Soundness of the probability-to-linear translation then lifts linear infeasibility to a proof that the original problem has no real model.

For the supported rational affine fragment, the backend is intended to be decisive: feasibility yields an exact rational model, and infeasibility yields a checked contradiction. No floating-point tolerance is used in either branch.

5.4 Nonlinear algebraization and probability-aware preprocessing

Constraints involving independence, mutual independence, products, and powers are translated to polynomial relations over the same atom variables. Conditional expressions are represented internally by polynomial numerators and denominators together with explicit positivity obligations; cross-multiplication is performed only after those obligations have been recorded. The Lean development proves a soundness theorem from models of the original probability problem to solutions of the generated polynomial system.

Before invoking a general nonlinear solver, PROBCONSISTENCY applies domain-specific reductions. A reduction is represented by a structure containing a reduced problem and a proof that every model of the original problem is a model of the reduced problem. Such reductions compose. Their logical use is intentionally one-sided:

$$\text{reduced problem inconsistent} \implies \text{original problem inconsistent},$$

whereas consistency of a necessary reduction alone does not establish consistency of the original problem.

Current reductions include exact consequences of pairwise independence when rational marginals or complement marginals are known, repeated processing of several independence assumptions, and scalar reductions for equiprobable mutually independent families. For example, the five-event gap problem P6 is reduced to

$$0 \leq p \leq 1, \quad \frac{27}{50} \leq p - p^5.$$

The reduction from the event-level assumptions to this scalar condition is proved in Lean. The scalar infeasibility test itself is presently delegated to Mathematica unless a separate Lean certificate is supplied.

5.5 External nonlinear search

For nonlinear cases not settled by certified preprocessing, the package can serialize the exact polynomial system to Wolfram Language and invoke a local Mathematica or Wolfram Engine process. Rational constants remain exact. Two uses of the external system are distinguished.

First, `FindInstance` may propose an atom assignment. When every coordinate is rational, Lean parses the candidate, checks its length, nonnegativity, and normalization, and re-evaluates every *original* assumption by exact rational arithmetic. Only a successful independent check produces `CONSISTENT`. Mathematica is therefore a search oracle in this branch, not a trusted verifier.

Second, Mathematica may report that the generated real-algebraic problem has no solution. The formal development proves the reduction supplied to Mathematica, but it does not yet reconstruct a general CAD or Positivstellensatz certificate from the external output. The outcome is therefore reported as `EXTERNAL_INFEASIBILITY_CLAIM`, not `INCONSISTENT`. Nonrational model candidates, malformed output, unavailable external software, and unresolved searches remain explicitly inconclusive.

5.6 The unified checking pipeline

The public command is

```
#prob_check problem.
```

At a high level, the current runner executes the following sequence:

1. attempt the certified exact linear backend;
2. recognize selected Lean-proved scalar reductions;
3. apply compositional probability-aware preprocessing and test the resulting necessary linear projection;
4. invoke full atom-level Mathematica model search as a fallback;
5. verify any returned rational model against the original problem.

The first conclusive certified branch terminates the search. External infeasibility remains marked as external, while a feasible scalar necessary condition falls through to the full model search because it does not by itself prove feasibility of the original problem.

This architecture operationalizes RQ3. The package does not ask a theorem prover to notice inconsistency incidentally while proving an unrelated target. It constructs a dedicated model-existence judgment, chooses a backend appropriate to the algebraic form of the premises, and records the evidence supporting the outcome.

5.7 Scope of the assurance

The package certifies the Lean object supplied to it. It does not certify that an informal statement was translated faithfully into that object. This limitation is especially important in an LLM-mediated workflow: autoformalization can omit an assumption, reverse an inequality, confuse pairwise and mutual independence, or exchange the target and conditioning events in a conditional probability. The formal input must therefore be reviewed independently, even when an LLM prepares the file (Wu, Jiang, Li, Rabe, Staats, Jamnik & Szegedy, 2022).

Within that boundary, PROBCONSISTENCY provides the assurance absent from the proof-assistant experiments of the preceding sections. Kernel checking still answers whether a proof term is valid relative to its context. PROBCONSISTENCY adds a distinct, explicit question: whether that context has a probability model. The implementation does not eliminate the difference between syntax and intended meaning, but it prevents model existence from being silently inferred from successful deduction. Technical definitions, module structure, and certificate paths are recorded in Appendix C.

6 Limitations

The study is deliberately bounded. It uses six constructed problem families, six model configurations, and one mathematical domain; the repeated trials do not support general rankings of LLMs or proof assistants. Coding used visible responses only, and the LLM and prover experiments were controlled component studies rather than a deployed closed-loop system. Proof-search results also depend on nonidentical encodings, tactics, prover portfolios, and resource limits.

The present results refer to the current available models; LLMs evolve might absorb the task of checking for consistency as a desirable feature.

The developed safeguard has corresponding limits. Atomization introduces 2^n masses, and both elimination and nonlinear procedures may scale poorly. PROBCONSISTENCY is a public alpha: the certified linear fragment is exact but restricted, certified consistency currently relies on exact rational witnesses, and general nonlinear infeasibility may remain external to Lean. Most importantly, the package checks the Lean object it receives, not whether an LLM translated the source text faithfully. Consistency is one necessary assurance layer, not a substitute for semantic review or proof checking.

7 Conclusion

This paper considered a concrete assurance gap in neurosymbolic reasoning. As expected, a proof assistant certifies that a conclusion follows from a formal context without weighing on the issue that the context describes any possible model, while LLMs alternate between local derivation and global premise checking according to how the task is framed. In the benchmark, we explored these issues in details for a fragment of finite probabilities, and while unprompted inconsistency detection was far from guaranteed, explicit premise-first prompting made inconsistency detection reliable with no observed completed-control false positive. Across Lean, jsCoq, and Isabelle, kernels accepted valid proofs from inconsistent contexts, whereas automated contradiction discovery remained selective and representation-dependent. This raises the possibility of misalignment with the human general understanding that, excluding explicitly identified cases, premises of a logical deduction are consistent.

To illustrate possible safeguards, we gave for finite elementary probability, an exact semantic account of the missing obligation by reducing premise consistency to semialgebraic feasibility over Boolean-atom masses. Finally, PROBCONSISTENCY turned this account into an executable Lean workflow for supported cases that keeps certified results, external claims, and inconclusive outcomes distinct.

In the broader context, reliable LLM–proof-assistant integration requires alignment not only between a proof term and a kernel, but also among the user’s intended scenario, the generated formal premises, and the interpretation communicated after checking. Premise satisfiability should therefore be an explicit, independently evidenced output of a neurosymbolic reasoning process. The present work aimed at raising a specific and increasingly important gap: a successful deduction can no longer silently stand in for the possibility of its premises.

Acknowledgements

[To be completed.]

Declaration of conflicting interests

The author declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

[To be completed.]

Data and code availability

The benchmark prompts, coded responses, proof-assistant files, analysis scripts, and the ProbConsistency Lean package will be made openly available in the project repository and an archival release. The final manuscript will provide persistent links and version identifiers.

References

- Azerbayev, Z., Schoelkopf, H., Paster, K., Dos Santos, M., McAleer, S., Jiang, A. Q., Deng, J., Biderman, S. & Welleck, S. (2024). Llemma: An open language model for mathematics. In *International Conference on Learning Representations*.
- Basu, S., Pollack, R. & Roy, M.-F. (2006). *Algorithms in Real Algebraic Geometry* (2 Ed.). Springer.
- Besold, T. R., d'Avila Garcez, A., Bader, S., Bowman, H., Domingos, P., Hitzler, P., K\"uhnberger, K.-U., Lamb, L. C., Lowd, D., Lima, P. M. V., de Penning, L., Pinkas, G., Poon, H. & Zaverucha, G. (2017). Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*.
- Canny, J. (1988). Some algebraic and geometric computations in pspace. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing* (pp. 460–467).
- Collins, G. E. (1975). Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Automata Theory and Formal Languages*, Volume 33 of *Lecture Notes in Computer Science* (pp. 134–183). Springer.
- da Costa, N. C. A. & French, S. (2003). *Science and Partial Truth: A Unitary Approach to Models and Scientific Reasoning*. Oxford: Oxford University Press.
- Egré, P. (2021). The logic of conditionals. In E. N. Zalta (Ed.), *The Stanford Encyclopedia of Philosophy* (Winter 2021 Ed.).
- Evans, J. S. B. T. & Over, D. E. (2004). *If: Supposition, Pragmatics, and Dual Processes*. Oxford: Oxford University Press.
- Fagin, R., Halpern, J. Y. & Megiddo, N. (1990). A logic for reasoning about probabilities. *Information and Computation*, 87(1–2), 78–128.

- Gandolfi, A. (2026). Probconsistency: A lean 4 package for finite probability-constraint consistency. Software repository. Public alpha release.
- Hansen, P. & Jaumard, B. (2000). Probabilistic satisfiability. In *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, Volume 5. Springer.
- Ibeling, D. & Icard, T. (2020). Probabilistic reasoning across the causal hierarchy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Volume 34 (pp. 10170–10177).
- de Moura, L., Kong, S., Avigad, J., van Doorn, F. & von Raumer, J. (2015). The lean theorem prover (system description). In *Automated Deduction – CADE-25*, Volume 9195 of *Lecture Notes in Computer Science* (pp. 378–388). Springer.
- Nilsson, N. J. (1986). Probabilistic logic. *Artificial Intelligence*, 28(1), 71–87.
- Stalnaker, R. C. (1975). Indicative conditionals. *Philosophia*, 5(3), 269–286.
- Tarski, A. (1951). *A Decision Method for Elementary Algebra and Geometry* (2 Ed.). Berkeley: University of California Press.
- Trinh, T. H., Wu, Y., Le, Q. V., He, H. & Luong, T. (2024). Solving olympiad geometry without human demonstrations. *Nature*, 625, 476–482.
- Wu, Y., Jiang, A. Q., Li, W., Rabe, M. N., Staats, C., Jamnik, M. & Szegedy, C. (2022). Autoformalization with large language models. *arXiv preprint arXiv:2205.12615*.
- Yang, K., Swope, A., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R. & Anandkumar, A. (2023). Leandojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems*, Volume 36.
- Zheng, K., Han, J. M. & Polu, S. (2021). Minif2f: A cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*.

APPENDIX

A Formal presentation of the finite-event probability fragment

This appendix records a deductive presentation corresponding to the semantics of Section 4. It is adapted to the problem-specific use of probability assumptions in this paper. Related sound and complete systems for linear and nonlinear probability languages are available in the literature, including a richer causal hierarchy (Fagin, Halpern & Megiddo, 1990; Ibeling & Icard, 2020).

A.1 Syntax

For a fixed $n \geq 0$, let

$$\mathbb{E}_n = \{E_1, \dots, E_n\}$$

be the primitive event symbols. Event expressions are generated by

$$B ::= E_i \mid \top \mid \perp \mid B^c \mid B \cap B \mid B \cup B.$$

Probability terms are generated by

$$t ::= q \mid P(B) \mid t + t \mid t - t \mid t t \mid t^k, \quad q \in \mathbb{Q}, \quad k \in \mathbb{N}.$$

Atomic probability formulas have the form $t_1 \bowtie t_2$, where $\bowtie \in \{=, \neq, <, \leq, >, \geq\}$, and formulas are closed under the usual Boolean connectives. Rational constants may equivalently be eliminated by clearing positive denominators.

Conditional probability and independence are definitional abbreviations. In particular,

$$P(B \mid C) = q$$

abbreviates $P(B \cap C) = qP(C)$ together with $P(C) > 0$, and pairwise independence abbreviates $P(B \cap C) = P(B)P(C)$. For finite mutual independence, the appropriate product equality is required for each nonempty subfamily.

A.2 Semantics

A marked probability model is a tuple

$$\mathcal{M} = (\Omega, \mathcal{F}, P; E_1^{\mathcal{M}}, \dots, E_n^{\mathcal{M}}),$$

where $(\Omega, \mathcal{F}, P)$ is a probability space and every marked event is measurable. Boolean operations on event expressions are interpreted as the corresponding set operations, probability terms as real numbers, and comparisons by the usual order on $\mathbb{R}$. Satisfaction $\mathcal{M} \models \varphi$ is then defined compositionally.

For a finite set of assumptions G , write

$$\mathcal{M} \models G$$

when $\mathcal{M}$ satisfies every member of G . The set G is consistent when such an $\mathcal{M}$ exists. Semantic consequence is defined in the standard way:

$$G \models \varphi \iff \text{every marked probability model of } G \text{ satisfies } \varphi.$$

This definition intentionally remains meaningful when G is inconsistent; in that case every formula is a semantic consequence. Model existence is therefore a separate judgment.

A.3 A sound and complete deductive presentation

Let FP_n be a proof system containing the following components.

1. Propositional tautologies and modus ponens.
2. Boolean-algebra reasoning for event expressions.
3. A sound and complete calculus for real closed fields, applied to polynomial equalities and inequalities over probability terms.
4. The probability schemata

$$\begin{aligned} P(B) &\geq 0, \\ P(\top) &= 1, \quad P(\perp) = 0, \\ P(B \cup C) + P(B \cap C) &= P(B) + P(C), \\ P(B) &= P(C) \quad \text{whenever } B \text{ and } C \text{ are Boolean-equivalent.} \end{aligned}$$

For a particular problem, the formulas in G are added as hypotheses rather than treated as universally valid probability axioms.

Soundness and completeness. For every finite $G \cup \{\varphi\}$ in the language over $E_1, \dots, E_n$,

$$G \vdash_{\text{FP}_n} \varphi \iff G \models \varphi.$$

Proof sketch. Soundness follows from the probability axioms, Boolean set identities, and the soundness of real closed-field reasoning. For completeness, introduce the 2^n atom variables y_α from Section 4.3. The probability schemata derive that each event probability equals its atom sum and that the atom variables satisfy the simplex constraints. The assertion $G \not\models \varphi$ is therefore equivalent to the satisfiability over the reals of the atom translation of $G \wedge \neg\varphi$. If no such real assignment exists, completeness of real closed-field reasoning yields a proof of its negation; translating the atom equalities back gives a derivation of φ from G . Conversely, a satisfying atom assignment constructs the finite countermodel described in the realization theorem. $\square$

This proof also clarifies the role of first-order logic. The user-facing probability formula is quantifier-free, but model existence introduces an existential quantifier over the real atom masses. The resulting sentence lies in the existential theory of a real closed field.

B Exact algebraizations of the benchmark pairs

The following reductions use exact rational arithmetic even when the prompts use terminating decimals. Each inconsistent problem is reduced to an empty semialgebraic set or an explicit algebraic contradiction; each control is accompanied by a finite witness. The notation $y_{i_1 \dots i_n}$ denotes the mass of the atom on which E_j has truth value i_j .

B.1 P1: complement normalization

Let $y_1 = P(A)$ and $y_0 = P(A^c)$. The inconsistent assumptions require

$$y_1 = \frac{3}{4}, \quad y_0 = \frac{2}{3}, \quad y_0 + y_1 = 1,$$

which is impossible. The control replaces $2/3$ by $1/4$; the two atom masses $y_1 = 3/4$ and $y_0 = 1/4$ form a probability model.

B.2 P2: independence and disjointness

Let

$$a = P(A), \quad b = P(B), \quad q = P(A \cap B).$$

The inconsistent instance requires

$$a = b = \frac{1}{2}, \quad q = ab, \quad q = 0.$$

Independence gives $q = 1/4$, whereas disjointness gives $q = 0$. In the control, disjointness is removed. Assigning mass $1/4$ to each of the four Boolean atoms gives independent events with both marginals equal to $1/2$.

B.3 P3: conditional decomposition and nonnegativity

The assumptions imply

$$P(A^c) = 0.3, \quad P(A^c \cap B) = 0.6 \cdot 0.3 = 0.18,$$

and hence

$$P(A \cap B) = 0.95 - 0.18 = 0.77.$$

The remaining atom inside A would then have mass

$$P(A \cap B^c) = P(A) - P(A \cap B) = 0.7 - 0.77 = -0.07,$$

contradicting nonnegativity.

For the control, $P(B)$ is changed to 0.47. The four atom masses

$$P(A \cap B) = 0.29, \quad P(A^c \cap B) = 0.18, \quad P(A \cap B^c) = 0.41, \quad P(A^c \cap B^c) = 0.12$$

are nonnegative, sum to one, and satisfy all control assumptions.

B.4 P4: independence and a forbidden atom

Independence of A_1 and A_3 implies independence of their complements, so

$$P(A_1^c \cap A_3^c) = 0.65^2 = 0.4225.$$

The zero-mass assumption for $A_1^c \cap A_2^c \cap A_3^c$ forces all this mass into $A_1^c \cap A_2 \cap A_3^c$. But this event is contained in A_2 , whose probability is 0.35. Thus the translated system requires the false inequality $0.4225 \leq 0.35$.

For the control, each marginal is 0.40. With atom order (A_1, A_2, A_3) , the assignment

$$y_{010} = 0.36, \quad y_{110} = 0.04, \quad y_{100} = 0.20, \quad y_{001} = 0.24, \quad y_{101} = 0.16,$$

with all other atom masses zero, is a witness. Its marginals are 0.40, $P(A_1 \cap A_3) = 0.16 = 0.40^2$, and $y_{000} = 0$.

B.5 P5: conditional probabilities on a partition

The partition and likelihood assumptions give

$$P(A_1 \cap O) = \frac{1}{6}, \quad P(A_2 \cap O) = \frac{1}{3}, \quad P(A_3 \cap O) = 0,$$

so $P(O) = 1/2$ and therefore

$$P(A_1 \mid O) = \frac{1}{3}.$$

Because A_1, A_2, A_3 are an equiprobable partition,

$$P(A_1 \mid A_3^c) = \frac{1}{2}.$$

The additional equality between these two conditional probabilities is thus inconsistent. In denominator-free polynomial form, it requires

$$P(A_1 \cap O)P(A_3^c) = P(A_1)P(O),$$

but the two sides evaluate to $1/9$ and $1/6$, respectively.

The control removes this additional equality. A finite witness is obtained by assigning the six nonempty partition cells

$$P(A_1 \cap O) = \frac{1}{6}, \quad P(A_1 \cap O^c) = \frac{1}{6}, \quad P(A_2 \cap O) = \frac{1}{3}, \quad P(A_2 \cap O^c) = 0,$$

$$P(A_3 \cap O) = 0, \quad P(A_3 \cap O^c) = \frac{1}{3}.$$

B.6 P6: a nonlinear mutual-independence gap

Let p be the common marginal. Mutual independence gives full-intersection probability p^5 , so the inconsistent threshold is

$$p - p^5 \geq \frac{27}{50}, \quad 0 \leq p \leq 1.$$

A rational polynomial certificate proves that this semialgebraic set is empty. Set $a = 2/3$. The identity

$$p^5 - 5a^4p + 4a^5 = (p - a)^2(p^3 + 2ap^2 + 3a^2p + 4a^3)$$

has a nonnegative right-hand side on $[0, 1]$. Hence

$$p - p^5 \leq \frac{p}{81} + \frac{128}{243} \leq \frac{131}{243} < \frac{27}{50},$$

contradicting the assumed gap.

The control threshold $53/100$ is feasible. Taking $p = 2/3$ gives

$$p - p^5 = \frac{130}{243} > \frac{53}{100}.$$

Five independent Bernoulli coordinate events with this common marginal form an explicit finite model.

B.7 Use in automated consistency checking

For each benchmark pair, the generic atom translation consists of the simplex constraints, the linear atom sums for all named event expressions, and the translated polynomial assumptions. A real quantifier-elimination procedure may then decide the existential formula directly. The reduced arguments above serve the same semantic purpose while exposing small certificates that can be checked independently by a proof assistant. The next section describes how PROBCONSISTENCY combines these two approaches and records the resulting trust boundary.

C Technical implementation details of PROBCONSISTENCY

This appendix records the implementation details needed to reproduce and audit the consistency layer described in Section 5. It complements the language-theoretic presentation in Appendix A and the benchmark-specific reductions in Appendix B; it does not repeat their semantic proofs. The source code is released under the Apache License 2.0 at

<https://github.com/agl89NYU/ProbConsistency>

The corresponding software citation is (Gandolfi, 2026). The repository's `lean-toolchain` and `lake-manifest.json` files pin the Lean toolchain and dependency revisions used by the public alpha.

C.1 Source organization

The implementation is divided into layers whose mathematical interfaces are narrower than their file boundaries. Table 7 lists the principal components.

Table 7. Principal implementation layers. File names are relative to the ProbConsistency/ source directory.

Layer and representative modules	Responsibility
Core syntax Core/EventExpr.lean Core/ProbExpr.lean Core/Constraint.lean Core/Problem.lean	Typed event expressions, probability expressions, labeled constraints, and the problem record.
Semantic layer Semantics/Atom.lean Semantics/AtomProbability.lean Semantics/PMFModel.lean Semantics/Satisfaction.lean	Boolean atoms, atom sums, normalized assignments, finite PMF realization, well-defined conditional probabilities, and satisfaction.
Certified linear layer Algebra/Linear.lean Algebra/LinearSoundness.lean Solver/ExactLinearFeasibility.lean Solver/ExactLinearBounds.lean Solver/ExactLinearModelCorrectness.lean	Affine translation, translation theorems, exact rational Fourier–Motzkin elimination, strict-bound handling, and model reconstruction.
Linear certificates and runner Solver/FourierMotzkinCertificate.s.lean Solver/StrictFourierMotzkinCertificates.lean Solver/LinearCertificates.lean Solver/LinearRunner.lean	Checked infeasibility certificates and production of proof-carrying public linear verdicts.
Polynomial layer Algebra/Polynomial.lean Algebra/PolynomialAlgebraization.lean Algebra/PolynomialSoundness.lean	Exact polynomial syntax, denominator-aware translation, and soundness of the nonlinear algebraization.
Preprocessing Algebra/ProbabilityPreprocess.lean	Composable model-preserving reductions for independence patterns, necessary linear projections, and scalar power-gap problems.
External backend Solver/MathematicaCADFormat.lean Solver/MathematicaCADBackend.lean Solver/MathematicaRationalModels.lean Solver/MathematicaScalarPowerGap.lean	Exact query serialization, process execution, output classification, rational-candidate parsing, and external scalar feasibility checks.
Orchestration and interface Solver/NonlinearRunner.lean Command/Check.lean	Trust-aware backend selection, user messages, printed atom models, and the #prob.check commands.

C.2 Core datatypes and atom semantics

For a fixed number n of primitive events, the Lean type of Boolean atoms is definitionally

$$\text{Atom } n = \text{Fin } n \rightarrow \text{Bool}.$$

The function `Atom.all n` computes every valuation exactly once. The development proves membership and no-duplication theorems and constructs an equivalence between list positions and atoms. This gives a canonical coordinate system for all subsequent algebra.

An atom-mass function over a coefficient domain R has the conceptual type

$$\text{Atom } n \rightarrow R.$$

An `AtomMassAssignment n R` packages such a function with nonnegativity and normalization proofs. The probability of an event expression is the finite sum

$$P_x(B) = \sum_{\alpha \in \Sigma_n} \mathbf{1}_{\alpha \models B} x_\alpha.$$

The semantics module defines evaluation of probability expressions and separately defines their well-definedness. For a conditional expression, well-definedness requires a strictly positive conditioning probability. A labeled constraint holds exactly when its underlying constraint holds; the label is semantically inert.

The PMF realization module turns a normalized nonnegative atom assignment into a finite probability mass function on Σ_n . It then proves that evaluation of a Boolean event by atom summation agrees with its PMF probability. Consequently, the executable atom semantics is linked to the ordinary finite probability semantics used in the realization theorem.

C.3 Executable rational verification

The simplest certified backend checks a supplied rational assignment. Each semantic predicate has a Boolean counterpart: well-definedness of probability expressions, rational comparisons, and all constraint constructors. Reflection lemmas prove equivalences of the form

$$\begin{aligned} \text{check assignment constraint} &= \text{true} \\ \iff \text{constraint.holds assignment.} \end{aligned}$$

List-level reflection then gives

$$\begin{aligned} \text{problem.checkRationalAssignment} \\ \text{assignment} &= \text{true} \\ \iff \text{problem.SatisfiedBy assignment.} \end{aligned}$$

A successful computation is converted to `RationalConsistencyCertificate problem`, whose two fields are the rational atom assignment and a proof that it satisfies the problem. A rejected assignment produces no negative conclusion: failure of one candidate is not evidence that all candidates fail. This checker is reused after internal linear reconstruction and after Mathematica proposes a rational model.

C.4 Affine translation

An affine expression is represented by a rational constant and a rational coefficient function on atoms:

$$c + \sum_{\alpha} a_{\alpha} x_{\alpha}.$$

The probability of a Boolean event is translated by assigning coefficient one to its satisfying atoms and zero to the others. The generated linear system always includes

$$x_{\alpha} \geq 0 \quad (\alpha \in \Sigma_n), \quad \sum_{\alpha} x_{\alpha} = 1.$$

The translator accepts exact rational constants, event probabilities, addition, subtraction, multiplication by a recognized constant, and the supported power cases. Relations are translated to affine comparisons. An equality $P(B \mid C) = q$ produces the pair

$$0 < P(C), \quad P(B \cap C) = qP(C).$$

The positivity constraint is part of the output rather than an implicit side condition.

The soundness development proves that a satisfying probability assignment satisfies all generated affine constraints. The reverse rational theorem shows that a rational atom assignment satisfying the translated system also satisfies the corresponding original constraints, including well-definedness of supported conditional probabilities. At problem level, the translation therefore supports both uses required by the solver:

$$\begin{aligned} \text{original model} &\implies \text{linear-system model}, \\ \text{rational linear-system model} &\implies \text{original rational model}. \end{aligned}$$

C.5 Exact Fourier–Motzkin decision and reconstruction

The linear decision layer uses rational inequalities of the form

$$a_1 x_1 + \cdots + a_m x_m \leq b \quad \text{or} \quad a_1 x_1 + \cdots + a_m x_m < b.$$

Equalities are represented by two weak inequalities. Each inequality stores a Boolean strictness flag. To eliminate x_1 , inequalities are separated according to whether the leading coefficient is positive, negative, or zero. Every positive–negative pair yields a projected inequality in the remaining coordinates. The projected inequality is strict whenever at least one member of the pair is strict.

After all variables have been eliminated, the residual system consists of constant inequalities. A false residual proves infeasibility. If every residual is true, reconstruction proceeds in reverse order. For a fixed tail vector, each original inequality induces a lower bound, an upper bound, or a residual condition on the eliminated coordinate. The implementation computes the strongest lower and upper bounds, proves pairwise compatibility from satisfaction of the projected system, and selects an exact rational coordinate. Iterating this argument reconstructs a complete rational vector.

The proof development establishes:

1. soundness of elimination;
2. preservation of coefficient-vector lengths;
3. soundness of each reconstructed coordinate;
4. completeness of reconstruction from a feasible residual;
5. agreement between vector coordinates and the `AtomMasses.ofValues` construction applied to `(Atom.all n)` values.

The final agreement theorem closes a subtle implementation gap: list-based linear algebra and function-based atom semantics must use the same atom ordering.

The public linear runner first translates the problem. A feasible exact decision is converted directly into a rational consistency certificate. An infeasible decision triggers reconstruction of a checked Fourier–Motzkin certificate, including the strict-residual case, and yields a proof of `¬problem.Satisfiable`. A malformed translation is reported as inconclusive.

C.6 Polynomial algebraization

The nonlinear expression language is a syntax tree generated by rational constants, atom variables, addition, negation, multiplication, and natural powers. Event probability is represented as the sum of the variables of the satisfying atoms. To translate conditional expressions without losing definedness information, the implementation uses a structure

(numerator, denominator, positive-denominator factors).

Arithmetic on fractions propagates the positivity obligations. A relation between two fractions is cross-multiplied only after constraints asserting strict positivity of all recorded denominator factors have been generated.

High-level constraints translate as follows:

- pairwise independence becomes $P(B \cap C) = P(B)P(C)$;
- mutual independence generates a product equality for each finite subcollection used by the package semantics;
- disjointness becomes zero probability of the intersection;
- a partition becomes full probability of the union plus pairwise zero-intersection equations;
- ordinary polynomial comparisons are translated compositionally.

The polynomial structural constraints are again atom nonnegativity and normalization.

The principal theorem needed by the external backend is reduction soundness:

$$\begin{aligned} &\text{problem.SatisfiedBy assignment} \\ &\implies \text{problem.toPolynomialSystem} \\ &\quad \text{.SatisfiedByReal assignment.mass.} \end{aligned}$$

Hence, if the generated polynomial system has no real solution, the original probability problem has no model. Conversely, a candidate returned by an external system is not accepted through this implication alone; it is checked directly against the original problem by the rational witness checker.

C.7 Proof-carrying preprocessing

A probability-aware reduction has type `SoundPreprocessing problem`. It contains a reduced problem and a proof

$$\forall x (x \models \text{problem} \implies x \models \text{reduced}).$$

Such reductions compose. They support the theorem

$$\neg \text{reduced.Satisfiable} \implies \neg \text{problem.Satisfiable}.$$

The reverse implication is intentionally absent.

The current implementation searches assumption lists for selected independence patterns rather than requiring a single fixed order. When exact marginals are known, it can replace independence by an exact intersection probability. It can use either a direct marginal or a complement marginal, iterate over several independence assumptions, and retain a necessary linear projection. Backtracking prevents an unusable early independence assumption from blocking a later reducible one.

For equiprobable mutually independent families, the package proves the necessary power identity for the full intersection. A specialized scalar reduction maps the event problem to

$$0 \leq p \leq 1, \quad \delta \leq p - p^m.$$

The mapping from every original model to a scalar solution is a Lean theorem. A scalar no-solution result may therefore refute the original problem, but the trust level of the conclusion still depends on how scalar infeasibility was established.

C.8 Mathematica interface and trust boundary

The external adapter serializes polynomial constraints using exact Wolfram Language rationals and a canonical variable list $\{y_0, \dots, y_{2^n-1}\}$. It can generate `Reduce`, `Resolve`, `FindInstance`, and `CylindricalDecomposition` queries. The process layer invokes `wolframscript` directly, records its exit status, standard output, and standard error, and classifies the response conservatively.

For model search, a wrapper asks Mathematica to print one tagged line:

```
NO_INSTANCE,
RATIONAL [ r0, ..., rk-1 ],
NONRATIONAL [ ... ].
```

Additional output lines are ignored unless they contain a recognized tag. A rational list is parsed to exact Lean rationals. The parser is not part of the logical trust base for `CONSISTENT`: even a wrongly parsed list can become conclusive only if exact Lean verification proves that the resulting assignment satisfies the original problem.

The no-instance branch has a different status. The current adapter does not import a general CAD decomposition certificate, a real-closed-field proof, or a Positivstellensatz certificate. Consequently, `NO_INSTANCE` and `False` remain external claims. They are not inhabitants of the Lean type

$$\neg \text{problem.Satisfiable}.$$

C.9 Unified runner

The operational logic of `#prob_check` may be summarized as follows:

1. Run the certified linear backend.
2. If the problem matches a proved scalar power-gap reduction, run the scalar external query.
3. Otherwise apply the compositional independence preprocessor and test the necessary linear projection.
4. If no certified branch concludes, run full polynomial `FindInstance`.
5. Verify a rational candidate against the original problem.
6. Preserve external infeasibility, nonrational candidates, malformed output, and backend failures as distinct noncertified outcomes.

The user command prints a human-readable explanation and, for a certified rational model, the exact mass of every Boolean atom. The lower-level result types remain available to Lean developments that need to consume the proof objects rather than their printed representations.

C.10 Reproduction and repository access

The public repository is

```
https://github.com/ag189NYU/ProbConsistency.
```

A local copy can be built by

```
git clone https://github.com/ag189NYU/ProbConsistency.git
cd ProbConsistency
lake build
```

The ordinary build does not require Mathematica. Deterministic tests are kept under `ProbConsistency/Tests/`; Mathematica-dependent examples are kept under `LiveTests/` and are run explicitly, for example:

```
lake lean LiveTests/AutomaticPowerGap.lean
```

The repository includes:

- `README.md`, containing the package overview and first examples;
- `FORMALIZING_PROBLEMS.md`, describing the input language;
- `MATHEMATICA_SETUP.md`, describing optional external-backend installation;
- `TRUST_MODEL.md`, documenting the certified and external result boundaries;
- the Lean source, permanent tests, and live examples.

A user file imports the public module and invokes the checker:

```
import ProbConsistency

-- event and assumption declarations

#prob_check problem
```

For archival reproducibility, the final paper should record a release tag or commit hash in addition to the repository URL. The software bibliography entry should include the corresponding version identifier or archived DOI.

C.11 Current limitations

The present implementation is a public alpha rather than a complete industrial decision environment. Its main limitations are:

1. atomization introduces 2^n variables for n primitive events;
2. Fourier–Motzkin elimination can exhibit severe intermediate growth;
3. probability-aware preprocessing recognizes selected syntactic patterns rather than all logically equivalent formulations;
4. certified consistency currently requires an exact rational model; nonrational algebraic candidates are not yet checked in Lean;
5. general nonlinear infeasibility reported by Mathematica is external;
6. users must install and license Mathematica or Wolfram Engine to use the optional nonlinear backend;
7. the package checks the formal Lean problem, not the correctness of an informal-to-formal translation.

Natural extensions include verified algebraic-number evaluation, import of checkable Positivstellensatz or sum-of-squares certificates, reconstruction of CAD certificates, broader normalization of probability expressions, and an LLM-facing structured input layer that separates autoformalization from the certified model-existence check.