Faults in our Formal Benchmarks

Pawan Sasanka Ammanamanchi
Eleuther AI
pawansasanka@gmail.com

Siddharth Bhat
Cambridge
siddharth.bhat@cl.cam.ac.uk

Abstract

Recent focus in LLM-assisted theorem proving has seen the rapid introduction of several benchmarks. Benchmarks in Lean are considered highly reliable because of the nature of proof assistants & formal languages. However, we question the quality of popular datasets and find that multiple popular benchmarks contain material defects. We discuss common issues that persist across datasets, such as omitted side conditions, misformalization, and incorrect/incomplete translations, through examples. We aim to establish that such defects occur across all datasets we examined, despite their prevalent use. Bad datasets pose a threat to the quality of evaluations in the field, and we propose some better dataset standards for such releases.

1 Introduction

Formal mathematics has experienced a meteoric rise in deep learning, particularly in the context of automated theorem proving, where models are trained to write proofs in systems like Lean. Recent models such as DeepSeek-Prover V2 [8], Goedel Prover 2 [5], and Kimina Prover [10] rely on benchmarks like miniF2F and ProofNet to demonstrate progress. Yet, benchmarks require constant attention. LLM benchmarks have been plagued by issues such as mislabeling, data contamination, and noisy data that obscure their true capabilities.

Benchmarking in Lean is considered reliable because it leverages formal verification; every claimed property must be accompanied by a machine-checked proof, eliminating the logical errors and hidden assumptions that plague traditional benchmarks. Although these provide mathematical certainty through formal verification, their reliability is ultimately limited by the accuracy of the specifications and the translational correctness from natural language. Crucially, theorem provers certify a given formalization, not that it faithfully captures the intended problem.

In this paper, we sample problems from widely used datasets and identify issues in all of them. We document these recurring issues that plague these datasets and propose dataset standards to establish more reliable evaluation in formal mathematics.

2 Popular Formal Math datasets

1. **miniF2F** [13]: 488 Olympiad-level problems (244 validation, 244 test), originally released in Lean 3.
2. **ProofNet** [1]: 371 undergraduate-level problems from textbooks (analysis, algebra, topology), released in Lean 3 with paired natural-language statements and proofs.
3. **FormalMath** [12]: 5,560 Lean 4 problems spanning fromOlympiad challenges to undergraduate-level theorems across algebra, applied mathematics, calculus, number theory, and discrete mathematics. They also release FormalMath Lite, a carefully selected subset of 425 problems (comprising 359 high school-level and 66 undergraduate-level problems).

Dataset	#Problems	Lean (orig.)	Some Issues
miniF2F	488	Lean 3	Multiple forks/ports; Version proliferation, Incomplete formalization, Type errors, Incomplete specification.
ProofNet	371	Lean 3	Version proliferation, underspecification and incomplete formalization.
FormalMath	5,560	Lean 4	Arithmetic errors
CombiBench	100	Lean 4	Incomplete specification and Incorrect Translation.
ProverBench	325	Lean 4	Wrong Problems, Incomplete Specification, Incorrect Translation

Table 1: Benchmarks referenced in this study (details deferred to §3).

4. **CombiBench** [6]: A benchmark comprising 100 combinatorial problems, each formalized in Lean 4 and paired with its corresponding informal statement.
5. **ProverBench** [8]: A benchmark dataset comprising 325 problems. 15 are formalized from AIME 24 and 25. The remaining 310 problems are drawn from curated textbook examples and educational tutorials.

We don't perform an exhaustive study of all the issues in these datasets. Our observations are based on compiling and reading the public releases/forks cited for each benchmark.

3 What Goes Wrong

Recurring issues in Formal math benchmarks face distinct challenges that go beyond the typical issues affecting NLP benchmarks. While traditional benchmarks grapple with annotation quality and label noise, formal mathematics introduces layers of complexity unique to the intersection of natural language, formal logic, and evolving proof assistants.

Version Drift & Dataset Maintenance Lean is a rapidly evolving language [2] with years of development still ahead. miniF2F was originally released in Lean 3, which is obsolete today and not in use. While changes in Lean 3 to Lean 4 might look subtle, given its evolving codebase, ways of expressing some mathematical objects has become easier or has changed over time. For example, in 1, When ProofNet was originally released, `Perfect` wasn't available so the authors used a subpar and incorrect approximation, but it was solved later in ProofNet#.

Problem Statement

Rudin Exercise 2.28: Prove that every closed set in a separable metric space is the union of a (possibly empty) perfect set and a set which is at most countable.

ProofNet (Incomplete & Incorrect)

```
theorem exercise_2_28 (X : Type*)
  [metric_space X] [separable_space X]
  (A : Set X) (hA : is_closed A) :
  ∃ P₁ P₂ : set X, A = P₁ ∪ P₂ ∧
  is_closed P₁ ∧ P₁ = {x | cluster_pt x
  (ℙ P₁)} ∧ set.countable P₂ := sorry
```

ProofNet# (Correct)

```
theorem exercise_2_28 (X : Type*)
  [MetricSpace X] [SeparableSpace X]
  (A : Set X) (hA : IsClosed A) :
  ∃ P₁ P₂ : set X, A = P₁ ∪ P₂ ∧
  Perfect P₁ ∧ Set.Countable P₂ := sorry
```

Figure 1: Example of evolving formalization: The original version only requires P_1 to be closed, while the correct formalization explicitly requires it to be perfect (closed with no isolated points).

Benchmarks are released for a specific Lean/mathlib snapshot, but Lean is a relatively new and evolving system. As APIs and tactics change, previously valid files break; quick "fixes" can also introduce subtle semantic changes. This points to the need for dataset maintenance in a world of evolving languages, which rarely happens if not at all in the world of deep learning. This also leads to issues like version proliferation which we discuss next.

Proliferation of Incompatible Versions. This version drift creates a cascade effect. Over time, multiple versions of miniF2F have been created, some of which are ports from Lean3 while others claim to fix errors. The original version released on github [1], another one introduced in Draft Sketch Prove[4] [2], a port in Lean4 maintained by Kaiyu Yang [3], another version introduced in Kimina prover which fixes some errors [10] [4] and a version introduced by Harmonic [3][5]. Many of these versions are one-off releases, and some issues persist across versions and persist into downstream forks. The Harmonic version is the best because of extensive corrections, but is not used by anyone because of its improper splits. Papers rarely specify which version of the dataset was used hence comparing across models is difficult.

A similar issue also arose with ProofNet, the original version was released on github [1], a port to Lean4 [9], a version introduced in DeepseekProver 1.5 [11] and an update version with corrections (in 118 problems out of 371) released as ProofNet# in [7].

Misformalization. The ways in which misformalizations happen are as follows:

1. Incomplete Specification: When translating from natural language to Lean, forgetting some properties of mathematical objects can lead to trivial or vacuous or even wrong questions. This also extends to missing conditions. For example in 2, the formalization is missing the properties of V where it should be finite dimensional. The existence of complements/kernel intersections depends on finite-dimensionality; otherwise $U \cap kerT = 0$ with $rangeT = T(U)$ need not hold.

ProofNet - Axler Exercise 3.8

Problem Statement:

Suppose that V is finite dimensional and that $T \in \mathcal{L}(V, W)$. Prove that there exists a subspace U of V such that $U \cap \text{null } T = \{0\}$ and range $T = \{Tu : u \in U\}$.

Lean Formalization:

```
theorem exercise_3_8 {F V W : Type*} [add_comm_group V]
  [add_comm_group W] [field F] [module F V] [module F W]
  (L : V →[F] W) :
  ∃ U : submodule F V, U ⊓ L.ker = ⊥ ∧
  linear_map.range L = range (dom_restrict L U):=
```

Figure 2: Axler Exercise 3.8: Missing finite dimension hypothesis in formalization

2. Incorrect Translation: When translating from natural language to lean not all mathematical arguments might naturally translate, and this will lead to errors.
3. Incomplete Translation: When translating from natural language to lean, you need to translate every part of the question and its conditions for correctness. In 3, while the original inequality is correctly translated, the equality determination is absent from the formalization, silently dropping half the problem.
4. Wrong Specification: When translating from natural language to lean, translating conditions, and variables in domains should be done accurately. For example, if a question says `For every positive natural number A`, the correct lean translation would be `a : N+`, or `(a : N) (ha: a > 0)`. Wrong or weaker specifications can have various problems.

[1]https://github.com/openai/miniF2F
[2]https://github.com/facebookresearch/miniF2F
[3]https://github.com/yangky11/miniF2F-lean4/tree/main
[4]https://huggingface.co/datasets/AI-MO/miniF2F_test
[5]https://github.com/Harmonic-ai/datasets

miniF2F - IMO1983 Problem 6

Problem: Let a, b and c be the lengths of the sides of a triangle. Prove that $a^2b(a-b)+b^2c(b-c)+c^2a(c-a) \geq 0$. Determine when equality occurs.

Lean:

```
theorem imo_1983_p6 (a b c : ℝ)
  (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)
  (h₁ : c < a + b) (h₂ : b < a + c) (h₃ : a < b + c) :
  0 ≤ a^2 * b * (a - b) + b^2 * c * (b - c) +
      c^2 * a * (c - a) := begin sorry end
```

Figure 3: IMO 1983 P6: Missing equality condition in formalization

This is not an exhaustive list of such errors but provides a decent understanding of how they may look. We document several such errors across datasets in the A.1. Each misformalization doesn't just add noise, it fundamentally changes what we're asking models to prove. It can turn stronger problems to weaker ones, and sometimes missing the domains of variables can lead to arithmetic errors like truncated substraction, division by zero etc.

Absence of Verified Solutions. The best way to find misformalizations and wrong questions is to formalize the answer to questions, but while solutions enables verification they also risk contamination. So benchmarks release theorem statements without proofs, hoping they're correct. In practice this prevents validation and can mask unsatisfiable statements or vacuous truths.

4 Lessons Learnt: Towards Standard Practices

Turn off auto-implicit Most benchmarks are created by auto-formalizing natural language with LLMs. When the LLM mistypes a variable name or forgets to define a type parameter, Lean's auto-implicit feature silently "fixes" it by adding implicit parameters. Lean's default autoImplicit automatically inserts hidden type parameters and universe levels for undeclared symbols, which is ergonomic for humans but risky for NL→Lean pipelines. The file type-checks, so nobody notices. The error will only surface later when someone tries to prove it and finds it's either trivial or impossible. Adding 'set_option autoImplicit false' makes these errors fail at formalization time instead of hiding them. This forces errors at formalization time and the file won't compile until you explicitly declare the intended domains and binders.

Basic Checkers: Be disciplined about types and common arithmetic traps Natural number subtraction in Lean truncates: '2 - 3 = 0', not '-1'. This silently changes problems. Division by zero returns a default value. In fields/division rings as modeled in mathlib, $inv\ 0 = 0$ and thus $a/0 = 0$. This totalization can silently trivialize goals unless the denominator is guarded by $h : b \neq 0$. A minimal checker pass should flag Nat subtraction, unguarded division, and unintended real coercions.

Avoid using Axioms Benchmarks must not introduce axioms or constants to stand in for problem statements. In Lean, writing axiom $h : \phi$ (or constant $h : \phi$) asserts a proof of ϕ exists; any solver can then "solve" the item by citing h, which collapses evaluation and obscures whether the statement is even satisfiable. We should therefore adopt a no-axioms rule for dataset files: problem statements are encoded as definitions of propositions, never as axioms.

Dataset Maintenance and Pinning When datasets are released, the lean version they are intended for should also be made clear, so it serves as a reference when being solved by a model in more recent lean versions. Regardless of if the dataset is autoformalized or human written, there isn't always a mathematical equivalent in Lean and it is an evolving language so some definitions and ease of translation will change over time.

Capture all requirements, not just the main claim. While human verification is the gold standard for identifying errors in formalization, it is costly and time consuming. LLM-as-a-judge could be explored to identify if a problem is capturing all the requirements in a given statement vs just the main claim.

References

[1] Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev, and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathematics, 2023.

[2] Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In *CADE*, 2015.

[3] harmonic Community. Harmonic minif2f performance. `https://harmonic.fun/news#blog-post-2`.

[4] Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *Submitted to The Eleventh International Conference on Learning Representations*, 2022.

[5] Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang, Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction, 2025.

[6] Junqi Liu, Xiaohan Lin, Jonas Bayer, Yael Dillies, Weijie Jiang, Xiaodan Liang, Roman Soletskyi, Haiming Wang, Yunzhou Xie, Beibei Xiong, Zhengfeng Yang, Jujian Zhang, Lihong Zhi, Jia Li, and Zhengying Liu. Combibench: Benchmarking llm capability for combinatorial mathematics, 2025.

[7] Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Improving autoformalization using type checking, 2025.

[8] Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang, Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025.

[9] Rahul Vishwakarma. Proofnet lean4. `https://github.com/rahul3613/ProofNet-lean4`.

[10] Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxcé, Bolton Bailey, Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu, Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani, Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Barroyer, Wen-Ding Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying Liu, and Jia Li. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning, 2025.

[11] Huajian Xin, Z. Z. Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search, 2024.

[12] Zhouliang Yu, Ruotian Peng, Keyi Ding, Yizhe Li, Zhongyuan Peng, Minghao Liu, Yifan Zhang, Yuan Zheng, Huajian Xin, Wenhao Huang, Yandong Wen, and Weiyang Liu. Formalmath: Benchmarking formal mathematical reasoning of large language models. *arXiv preprint arXiv:2505.02735*, 2025.

[13] Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022.

A Issues in Formal Mathematics Datasets

This appendix has examples of various formalization issues found across major formal mathematics benchmarks. We present side-by-side comparisons of problematic formalizations, and describe the error and either point out which version is correct or provide the correction. This section is meant to familiarize the reader with the kinds of pitfalls we observe in datasets. This is not a complete list of all errors in datasets like miniF2F, ProofNet, FormalMath, Proverbench which would span 100s of problems if not 1000s.

A.1 Issue 1: Incomplete Formalization

Many problems in formal benchmarks only formalize part of the original problem, often missing crucial constraints or conditions.

A.1.1 IMO 1983 Problem 6 - miniF2F

In this problem, the solution must first prove the inequality and then determine when equality occurs.

Problem Statement

Let a, b and c be the lengths of the sides of a triangle. Prove that

$$a^2b(a-b)+b^2c(b-c)+c^2a(c-a)\geq 0.$$

Determine when equality occurs.

DSP/OpenAI/Numina Version

```
theorem imo_1983_p6
  (a b c : ℝ)
  (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)
  (h₁ : c < a + b)
  (h₂ : b < a + c)
  (h₃ : a < b + c) :
  0 ≤ a^2 * b * (a - b) +
      b^2 * c * (b - c) +
      c^2 * a * (c - a) :=
begin
  sorry
end
```

Harmonic Version (Correct)

```
theorem formal_1647
  (a b c d : ℝ)
  (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)
  (h₁ : c < a + b)
  (h₂ : b < a + c)
  (h₃ : a < b + c)
  (h₄ : d = a^2 * b * (a - b) +
            b^2 * c * (b - c) +
            c^2 * a * (c - a)) :
  0 ≤ d ∧ (d = 0 ↔ (a = b ∧ b =
    c)) := by
  sorry
```

A.1.2 IMO 1981 Problem 6 - miniF2F

The problem asks to "Determine $f(4,1981)$" - a specific computational problem requiring finding an actual value. While the formalization asks for a recurrence relation $f(4,y+1)=2^{f(4,y)+3}-3$. While it might be the case that the recurrence relation might be used to solve the computational problem, this is not an exact formalization. Proving the recurrence is necessary but not sufficient for computing the numerical value.

Problem Statement

The function $f(x,y)$ satisfies:

1. $f(0,y)=y+1$,
2. $f(x+1,0)=f(x,1)$,
3. $f(x+1,y+1)=f(x,f(x+1,y))$,

for all non-negative integers x,y. Determine $f(4,1981)$.

DSP version

```
theorem imo_1981_p6
  (f : ℕ → ℕ → ℕ)
  (h₀ : ∀ y, f 0 y =
    y + 1)
  (h₁ : ∀ x, f (x + 1)
     0 = f x 1)
  (h₂ : ∀ x y, f (x +
     1) (y + 1) =
       f x (f (x + 1)
     y)) :
  ∀ y, f 4 (y + 1) =
  2^(f 4 y + 3) - 3
     :=
begin
  sorry
end
```

Harmonic version

```
theorem formal_1514
  (f : ℕ → ℕ → ℕ)
  (h₀ : ∀ y, f 0 y =
    y + 1)
  (h₁ : ∀ x, f (x + 1)
     0 = f x 1)
  (h₂ : ∀ x y, f (x +
     1) (y + 1) =
       f x (f (x + 1)
     y)) :
  ∀ y, f 4 (y + 1) =
  2^(f 4 y + 3) - 3
     := by
  sorry
```

OpenAI version

```
theorem imo_1981_p6
  (f : ℕ → ℕ → ℕ)
  (h₀ : ∀ y, f 0 y =
    y + 1)
  (h₁ : ∀ x, f (x + 1)
     0 = f x 1)
  (h₂ : ∀ x y, f (x +
     1) (y + 1) =
       f x (f (x + 1)
     y)) :
  ∀ y, f 4 (y + 1) =
  2^(f 4 y + 3) - 3
     :=
begin
  sorry
end
```

A.2 Issue: Missing Specification

A.2.1 IMO 1962 Problem 2 - miniF2F

The DSP/numina version doesn't specify that $\sqrt{3-x} - \sqrt{x+1} > 0$, and while it might be perhaps immaterial while solving the problem, it is a case of missing specification and leaving nothing to ambiguity.

Problem Statement

Show that if the real number x satisfies the inequality $\sqrt{\sqrt{3-x}-\sqrt{x+1}} < \frac{1}{2}$, then $-1 \leq x < 1 - \frac{\sqrt{127}}{32}$.

DSP/Numina version

```
theorem imo_1962_p2
  (x : ℝ)
  (hₓ : 0 ≤ 3 - x)
  (h₁ : 0 ≤ x + 1)
  (h₂ : 1 / 2 < real.sqrt (3 - x) -
       real.sqrt (x + 1)) :
  -1 ≤ x ∧ x < 1 -
  real.sqrt 31 / 8 :=
begin
  sorry
end
```

Harmonic version

```
theorem formal_4518
  (x : ℝ)
  (hₓ : x ≤ 3)
  (h₁ : -1 ≤ x)
  (h₂ : 0 ≤ Real.sqrt (3 - x) -
       Real.sqrt (x + 1))
  (h₃ : Real.sqrt (Real.sqrt (3 - x
     ) -
       Real.sqrt (x + 1)) > 1 / 2)
     :
  -1 ≤ x ∧ x < 1 -
  Real.sqrt 127 / 32 := by
  sorry
```

A.2.2 APMO 1991 Problem 2 - CombiBench

In this problem originally, the points were not assumed to be distinct, so it was trivially false. It was fixed in the second version.

Problem Statement

Suppose there are 997 points given in a plane. If every two points are joined by a line segment with its midpoint coloured in red, show that there are at least 1991 red points in the plane.

Version 1

```
import Mathlib

noncomputable def red_points {k}
  (points : Fin k → ℝ × ℝ) :
  Finset (ℝ × ℝ) :=
  ((Finset.univ (α := Fin k × Fin
     k)).image (fun x => midpoint
     ℝ (points x.1) (points x.2))
    )

theorem apmo_1991_p2 (points : Fin
     997 → ℝ × ℝ) :
  (red_points points).card ≥ 1991
     := by sorry
```

Fixed

```
import Mathlib

noncomputable def red_points {k}
  (points : Fin k → ℝ × ℝ) :
  Finset (ℝ × ℝ) :=
(((Finset.univ (α := Fin k × Fin
    k)) \ (Finset.univ).image (
    fun i => (i, i))).image
    (fun x => midpoint ℝ (points
    x.1) (points x.2)))

theorem apmo_1991_p2 (points : Fin
     997 → ℝ × ℝ)
  (hpoints : Function.Injective
    points) :
  (red_points points).card ≥ 1991
     := by sorry
```

A.3 Issue: Incorrect Translation

Some formalizations fundamentally misrepresent the original problem, leading to different mathematical statements.

A.3.1 MATHD Algebra 15 - miniF2F

In this case, Harmonic version doesn't capture the general operation definition from the problem statement instead it directly introduces the substituted weaker version of the problem to solve. DSP and OpenAI versions define a general operation s while Harmonic directly computes the specific case.

Problem Statement

If $a * b = a^b + b^a$, for all positive integer values of a and b, show that $2 * 6 = 100$

DSP Version

```
theorem
    mathd_algebra_15
  (s : ℕ → ℕ → ℕ)
  (h₀ : ∀ a b, 0 < a
     ∧ 0 < b →
   s a b = a^(b:ℕ) +
     b^(a:ℕ)) :
  s 2 6 = 100 :=
begin
  rw h₀,
  refl,
  norm_num,
end
```

Harmonic Version

```
theorem formal_2895 :
  2 ^ 6 + 6 ^ 2 =
    100 := by
  sorry
```

OpenAI Version

```
theorem
    mathd_algebra_15
  (s : ℕ → ℕ → ℕ)
  (h₀ : ∀ a b, 0 < a
     ∧ 0 < b →
   s a b = a^(b:ℕ) +
     b^(a:ℕ)) :
  s 2 6 = 100 :=
begin
  rw h₀,
  refl,
  norm_num,
end
```

A.3.2 Number Theory Problem 16 - ProverBench

In this, problem u is a free variable and the specification that u is a substitute for $y - 2x$ is missing. Not only that, upon further look there might be an error in the problem, where $x + u\sqrt{3}$ does not solve the question, and the actual problem should have $u + x\sqrt{3}$.

Problem Statement

For the equation $x^2 + y^2 - 1 = 4xy$ its general solution in the integers is given by $x + u\sqrt{3} = (2 + \sqrt{3})^n$, where u is the substitute for $y - 2x$.

ProverBench Formalization (Incorrect)

```
import Mathlib

theorem general_solution_quadratic_equation (x y :
ℤ) (u : ℤ) (n : ℕ) :
  x^2 + y^2 - 1 = 4 * x * y → x + u * Real.sqrt 3 =
  (2 + Real.sqrt 3)^n :=
  sorry
```

A.4 Issue: Wrong Specification

A.4.1 quantitative_reasoning_zh_blue_41 - FormalMath Problem 5164

In this problem, truncated subtraction occurs due to natural numbers, and this is incorrect typecasting.

Problem Statement

Given $M = \{x \mid x = a^2 + 1, a \in \mathbb{N}^*\}$, $N = \{x \mid x = b^2 - 4b + 5, b \in \mathbb{N}^*\}$, then the relationship between M and N is Proof: The answer is $M \subseteq N$. From $a^2 + 1 = (a + 2)^2 - 4a + 5$, we can see that $M \subseteq N$. However, $1 \in N$ and $1 \notin M$.

FormalMath 5164

```
import Mathlib

open Set Real
open scoped BigOperators

theorem quantitative_reasoning_zh_blue_41 :
  {x : ℕ | ∃ a : ℕ, 0 < a ∧ x = a^2 + 1} ⊂ {x : ℕ |
  ∃ b : ℕ, 0 < b ∧ x = b^2 - 4 * b + 5} := by
  sorry
```

A.4.2 Problem 48 - FormalMath

This does not avoid the case that n can take the value 0.

Problem Statement

If the sequence $\{a_n\}$ satisfies that for any $n \in \mathbb{N}^*$, $\sum_{d|n} a_d = 2^n$, prove that $n \mid a_n$.

Lean Formalization:

```
theorem algebra_56552 (a : ℕ → ℕ) (ha : ∀ n, ∑ d in
  n.divisors, a d = 2 ^ n) :
  ∀ n, n ∣ a n := by
  sorry
```