# Decoupling Reasoning from Proving: A New Framework for Tackling Olympiad-Level Mathematics

**Zhenwen Liang**[1,*] **Linfeng Song**[1], **Yang Li**[2], **Tao Yang**[2], **Feng Zhang**[2], **Haitao Mi**[1], **Dong Yu**[1]

[1]Tencent AI Lab [2]Tencent LLM Department

## Abstract

Automated Theorem Proving (ATP) in formal languages is a foundational challenge for AI. While Large Language Models (LLMs) have driven remarkable progress, a significant gap remains between their powerful informal reasoning capabilities and their weak formal proving performance. Recent studies show that the informal accuracy exceeds 80% while formal success remains below 8% on benchmarks like PutnamBench. We argue this gap persists because current state-of-the-art provers, by tightly coupling reasoning and proving, are trained with paradigms that inadvertently punish deep reasoning in favor of shallow, tactic-based strategies. To bridge this fundamental gap, we propose a novel framework that decouples high-level reasoning from low-level proof generation. Our approach utilizes two distinct, specialized models: a powerful, general-purpose *Reasoner* to generate diverse, strategic subgoal lemmas, and an efficient *Prover* to rigorously verify them. This modular design liberates the model's full reasoning potential and bypasses the pitfalls of end-to-end training. We evaluate our method on a challenging set of post-2000 IMO problems, a problem set on which no prior open-source prover has reported success. Our decoupled framework successfully solves 5 of these problems, demonstrating a significant step towards automated reasoning on exceptionally difficult mathematical challenges. To foster future research, we will release our full dataset of generated and verified lemmas for a wide range of IMO problems, after the paper acceptance.

## 1 Introduction

Automated Theorem Proving (ATP) aims to automatically generate machine-verified proofs for mathematical statements. Recent progress, driven by Large Language Models (LLMs), has been substantial. However, a critical gap has emerged: top-tier LLMs can achieve over 80% accuracy in generating informal, natural-language solutions to complex math problems, but state-of-the-art formal provers struggle to solve even 8% of the same problems on benchmarks like PutnamBench [Dekoninck et al., 2025]. This highlights that while LLMs possess powerful mathematical reasoning abilities, current ATP systems fail to harness them for formal verification.

We argue this failure stems from a fundamental design flaw in modern provers like DeepSeek-Prover-v2 [Ren et al., 2025] and Kimina [Wang et al., 2025]. These models tightly couple high-level reasoning (planning or sketching) and low-level proof generation within a single, monolithic architecture. They are typically trained using Reinforcement Learning with Verifiable Rewards (RLVR), a paradigm that rewards only the final binary success of the generated code. This training objective inadvertently incentivizes models to suppress deep, human-like reasoning in favor of shallow, brittle strategies, such as brute-forcing automated tactics (`ring`, `omega`, etc.). This "reasoning degradation" explains their failure on exceptionally difficult problems, like International Mathematical Olympiad (IMO).

*Contact: zhenwzliang@global.tencent.com

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: MATH-AI.

We identify the root cause of this failure in the prevailing training paradigm: reinforcement learning with verifiable rewards (RLVR). This methodology, used to train models like DeepSeek-Prover-v2 and Kimina, rewards only the final binary success or failure of the generated Lean code. This paradigm is fundamentally misaligned with the goal of bridging the reasoning-proving gap. Instead of rewarding hard-to-define, human-like strategies (the kind that achieve >80% informal success), RLVR teaches a degenerated policy to maximize reward by any means necessary. It is incentivized to suppress its powerful, latent reasoning abilities in favor of heuristically decomposing goals into trivial sub-problems that can be solved by brute-forcing automatic tactics like ring or omega. This over-reliance is not merely a shortcut, but a symptom of a training-induced degradation of its reasoning capabilities.

To bridge this gap, we propose a new framework built on the principle of decoupling reasoning from proving. Our approach uses two distinct, specialized models: A powerful, general-purpose LLM as a dedicated Reasoner, tasked with generating high-level, strategic subgoal lemmas. An efficient, specialized ATP model as a Prover, tasked with formally verifying these lemmas and constructing the final proof. This architectural separation liberates the Reasoner to leverage its full reasoning capacity without being constrained by the immediate demands of formal proof generation. We evaluate our framework on a challenging set of post-2000 IMO problems and successfully solve 5 problems, a first for any open-source prover. To support future research, we release our dataset of verified lemmas for a wide range of IMO problems.

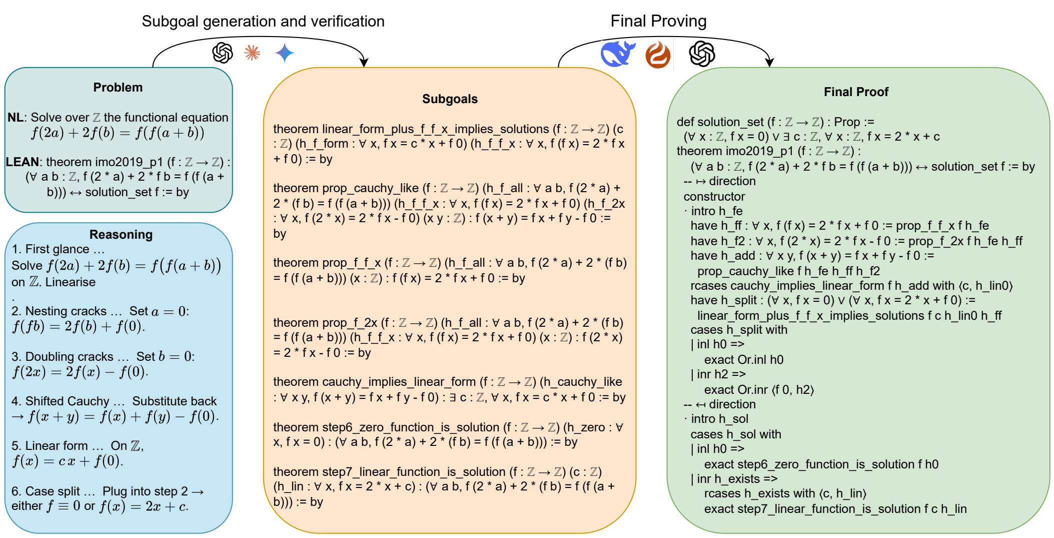


Figure 1: The overall pipeline of our framework, taking IMO 2019 P1 as an example. The Reasoner first generates strategic subgoals (lemmas). These are then verified by the Prover. Finally, the main theorem is proven using the verified lemmas as building blocks.

## 2 A Framework for Decoupled Reasoning and Proving

Our framework, illustrated in Figure 1, consists of a three-stage pipeline designed to leverage the distinct strengths of reasoning and proving models.

**Stage 1: Strategic Subgoal Generation.** The first stage uses a powerful, general-purpose LLM as a **Reasoner** (we found Gemini 2.5 Pro to be most effective) to decompose the main theorem into strategic subgoals. The quality of these subgoals is paramount, so we guide the Reasoner with a carefully designed prompt. The prompt instructs the model to first "think step-by-step to devise a feasible and complete proof strategy" in natural language *before* outputting any formal code. This forces it to engage in high-level planning. Crucially, it is also explicitly instructed to "avoid trivial splits" and ensure that each proposed subgoal represents a "meaningful proof milestone." This prevents the generation of shallow, unhelpful decompositions. Only after formulating a coherent plan is the model asked to translate these milestones into a list of formal Lean statements (e.g., ‘$theoremlemma_1 ... := bysorry$‘). This focuses the model on its core strength—strategic thinking—while constraining the output to a machine-parsable format, which we reliably extract using a simple regular expression.

**Stage 2: Subgoal Verification and Filtering.** Each candidate lemma from Stage 1 is passed to a dedicated Prover (we use DeepSeek-Prover-v2 7B) for verification. The Prover attempts to find a formal proof for each lemma. Only the lemmas that can be successfully proven are retained. This stage acts as a critical filter, grounding the Reasoner's creative and sometimes speculative ideas in formal logic.

**Stage 3: Final Proof Construction.** In the final stage, the prover attempts to solve the main theorem using the set of verified lemmas from Stage 2. A crucial challenge we identified here was domain shift. We initially hypothesized that the same Prover from Stage 2 (DeepSeek-Prover-v2) would be optimal. However, we observed that this model, when presented with auxiliary lemmas, often struggled to effectively utilize them, likely because its training data did not emphasize this specific pattern of formal proving. It tended to ignore the provided lemmas and attempt to prove the theorem from scratch, defeating the purpose of our pipeline. This led to a key insight: the ability to leverage existing lemmas is a distinct skill. After experimentation, we found that powerful general LLMs, such as OpenAI-o3 and Gemini 2.5 Pro, are significantly more adept at integrating and applying provided lemmas. This highlights the importance of selecting the right tool for each sub-task in a decoupled system.

# 3 Experiments and Analysis

We evaluated our framework on non-geometry problems from the International Mathematical Olympiad (IMO) from 2000 to 2024, a benchmark known for its extreme difficulty. Our method successfully solved 5 of these problems, for which no prior open-source prover had reported success: IMO 2000 Problem 2 ,IMO 2005 Problem 3, IMO 2011 Problem 3, IMO 2019 Problem 1, IMO 2020 Problem 2. We will release all the sub-theorems and full solutions upon paper acceptance.

## 3.1 Qualitative Analysis: Strategic vs. Brittle Reasoning

To illustrate the benefit of our approach, we compare the reasoning strategy generated by our framework for **IMO 2019 Problem 1** against that of the state-of-the-art DeepSeek-Prover-v2 671B.

Our **Reasoner** produced a structured, insightful decomposition that mirrors a human mathematician's approach. It began by identifying fundamental algebraic properties of the function (e.g., $f(f(x)) = 2f(x)+f(0)$), then used these to derive a crucial Cauchy-like additive identity. This identity logically led to the conclusion that the function must have a linear form ($f(x) = cx + d$), which it then used to constrain the parameters and find the two valid solutions. This is a hierarchical, property-driven strategy where each lemma builds logically on the last.

In stark contrast, **DeepSeek-Prover-v2** exhibited degenerate, brittle strategies shaped by its training objective. Its attempts were not strategic but tactical. One common failure mode was **brute-force instantiation**: generating a flat, unstructured list of dozens of ad-hoc algebraic identities by substituting various integers into the original equation (e.g., $f(0)$, $f(1)$, $f(x+1)$), and then hoping powerful automated tactics like `linarith` or `ring` could mechanically find a solution in the resulting morass of equations. Another typical strategy was **unjustified guessing**: asserting the final linear form of the solution as a hypothesis without any preceding logical derivation, and then attempting to work backward. These strategies lack mathematical insight and fail to scale, revealing a core weakness of models trained with end-to-end RLVR which rewards any path to a proof, no matter how unprincipled.

```
theorem amc12b_2002_p7 (a b c : ℕ) (h₀ : 0 < a ∧ 0 < b ∧ 0 < c)
(h₁ : b = a + 1) (h₂ : c = b + 1)
(h₃ : a * b * c = 8 * (a + b + c)) : a^2 + b^2 + c^2 = 77 := by
have hb : b = a + 1 := h₁
have hc : c = a + 2 := by omega
rw [hb, hc] at h₃
have h4 : a = 4 := by
```

```
have h_pos : 0 < a := h₀.left
have : a ≤ 6 := by
nlinarith
interval_cases a <;> omega
have ha : a = 4 := h4
have hb' : b = 5 := by omega
have hc' : c = 6 := by omega
rw [ha, hb', hc']
norm_num
```

### 3.2 Quantitative Evidence of Reasoning Degradation

Our central hypothesis is that RLVR-based fine-tuning for formal proving degrades a model's general mathematical reasoning ability. To test this, we compared the performance of a specialized prover (`Kimina-Prover`) with its general-purpose base model (`Qwen2.5-Math-7B-Instruct`) on standard math reasoning benchmarks (MATH and AIME) that do not require formal proof generation. The results in Table 1 provide clear evidence for our hypothesis. The prover model shows a significant and consistent performance drop across all metrics compared to its base model. This confirms that the specialization process for formal theorem proving, driven by verifiable rewards, comes at the cost of the model's intrinsic reasoning skills. This finding strongly motivates our decoupled approach, which preserves the full power of a dedicated reasoning model.

Table 1: Performance comparison on general mathematical reasoning benchmarks.

| | **MATH** | **AIME24** | | | |
|---|---|---|---|---|---|
| **Model** | **pass@1** | **pass@1** | **pass@4** | **pass@8** | **pass@16** |
| `Qwen2.5-Math-7B-Instruct` (base model) | **83.6%** | **16.7%** | **33.3%** | **43.3%** | **46.7%** |
| `Kimina-Prover-Preview-Distill-7B` (prover) | 78.7% | 11.0% | 24.1% | 32.0% | 40.9% |
| **Performance Drop (pts)** | **-4.9** | **-5.7** | **-9.2** | **-11.3** | **-5.8** |

### 3.3 Discussion

**On the Utilization of External Knowledge.** A key challenge we identified is that state-of-the-art provers, when presented with verified subgoals as standalone lemmas, often ignore them and attempt to re-prove everything from scratch. This "contextual blindness" suggests their training biases them against leveraging modular, pre-proven knowledge. In contrast, in-proof have statements force local fact usage but lack the flexibility of reusable lemmas. Our finding that general LLMs are better at utilizing external lemmas highlights a crucial gap: provers must be trained not just to prove, but to effectively continue proofs using existing results.

**Limitations and Failure Analysis.** Our framework's primary bottleneck is the Prover's inability to verify highly complex lemmas generated by the Reasoner. In an oracle setting where we manually proved these key lemmas, our framework could solve many more problems. This shows our pipeline is currently bounded by the raw power of the Prover component. A second, deeper challenge is the "ingenuity gap": our Reasoner excels at systematic, logical decomposition but struggles to produce the single, non-obvious "magical" insight that often characterizes elegant human solutions to the hardest problems.

## 4 Conclusion

In conclusion, we introduced a novel framework that decouples strategic reasoning from formal proof generation. By delegating high-level thinking to a powerful Reasoner and verification to a specialized Prover, our method successfully solves 5 post-2000 IMO problems, a new milestone for open-source ATP. Our analysis shows this separation is crucial for overcoming the reasoning degradation induced by current training paradigms. Future work will focus on improving the Prover's ability to handle complex lemmas and fine-tuning models to utilize pre-proven results.

## References


Jasper Dekoninck, Ivo Petrov, Kristian Minchev, Mislav Balunovic, Martin Vechev, Miroslav Marinov, Maria Drencheva, Lyuba Konova, Milen Shumanov, Kaloyan Tsvetkov, et al. The open proof corpus: A large-scale study of llm-generated mathematical proofs. *arXiv preprint arXiv:2506.21621*, 2025.

Emily First, Markus N Rabe, Talia Ringer, and Yuriy Brun. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1229–1241, 2023.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothee Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=SMa9EAovKMC`.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng YANG, Jing Tang, Jian Yin, Zhenguo Li, and Xiaodan Liang. Proving theorems recursively. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024a. URL `https://openreview.net/forum?id=yAa5l92TtQ`.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. LEGO-prover: Neural theorem proving with growing libraries. In *The Twelfth International Conference on Learning Representations*, 2024b. URL `https://openreview.net/forum?id=3f5PALef5B`.

Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, et al. Kimina-prover preview: Towards large formal reasoning models with reinforcement learning. *arXiv preprint arXiv:2504.11354*, 2025.

Xueliang Zhao, Lin Zheng, Haige Bo, Changran Hu, Urmish Thakker, and Lingpeng Kong. Subgoalxl: Subgoal-based expert learning for theorem proving. *arXiv preprint arXiv:2408.11172*, 2024.


## A Data Release

To foster further research and collaboration within both the mathematics and ATP communities, we are releasing a comprehensive dataset and a project website. While our framework successfully solved 5 IMO problems, our efforts in subgoal generation and verification have yielded a much larger collection of high-quality, formally verified lemmas for a broad range of post-2000 IMO problems. We believe this resource serves a dual purpose:

- For mathematicians and IMO researchers, this collection of machine-generated lemmas may offer novel perspectives or reveal non-obvious decompositions, potentially inspiring new human-led proof strategies.
- For the ATP community, our dataset acts as a new, challenging benchmark. By providing verified intermediate steps, it allows researchers to focus on solving the remaining difficult lemmas or on the final, complex proof-synthesis stage for problems currently beyond reach.

The dataset will be publicly available on HuggingFace, and we are committed to its active maintenance and expansion. We welcome community contributions, such as new proofs for existing lemmas or alternative strategic decompositions. The project website, which provides access to the data repository, tracks our ongoing progress, and presents detailed case studies, will be released after paper acceptance.

## B Comparing Reasoning Quality in IMO 2019 Problem 1

To evaluate the qualitative difference between reasoning strategies, we analyze how our framework's **Reasoner** compares against existing prover-driven approaches when applied to a challenging benchmark: IMO 2019 Problem 1. This problem asks to find all functions $f : \mathbb{Z} \to \mathbb{Z}$ satisfying $f(2a) + 2f(b) = f(f(a+b))$ for all integers $a, b$.

Our goal is to demonstrate that the reasoning path generated by our decoupled Reasoner-Prover framework leads to a principled, structured solution strategy, in stark contrast to prover-only models, which often exhibit brittle or degenerate behavior.

### B.1 Our Reasoner's Strategic Decomposition

In our framework, the Reasoner is responsible for identifying high-level mathematical structure and generating a roadmap of lemmas. On this problem, the Reasoner produces the following structured decomposition:

1. **Identify fundamental properties**: By strategic instantiation of the functional equation, the Reasoner isolates key identities:
   - `prop_f_f_x`: $f(f(x)) = 2f(x) + f(0)$ for all $x$
   - `prop_f_2x`: $f(2x) = 2f(x) - f(0)$ for all $x$
2. **Uncover additive structure**: Combining the above, the Reasoner deduces:
   - `prop_cauchy_like`: $f(x+y) = f(x) + f(y) - f(0)$
3. **Characterize the function form**: Using the Cauchy-like identity, the Reasoner infers:
   - `cauchy_implies_linear_form`: There exists $c \in \mathbb{Z}$ such that $f(x) = cx + f(0)$
4. **Constrain the parameters**: Plugging this linear form into `prop_f_f_x`, the Reasoner derives:
   - `linear_form_plus_f_f_x_implies_solutions`: $c$ must be either 0 or 2
5. **Verify candidate solutions**: Both resulting forms, $f(x) = 0$ and $f(x) = 2x + c$, are verified to satisfy the original equation.

This decomposition exhibits genuine mathematical insight: it identifies the functional equation's additive structure, abstracts useful intermediate results, and uses them to constrain the solution space efficiently and interpretably.

### B.2 The Decomposition Strategy by DeepSeek-Prover-v2 671B

We contrast this with the behavior of current state-of-the-art prover models. Specifically, we sampled three solution attempts from the strongest publicly available model, DeepSeek Prover v2 671B [Ren et al., 2025]. These are representative of the general behavior we observed. For brevity, we include only partial code excerpts.

The first attempt relies on a brute-force enumeration of equations. The model instantiates the functional equation on dozens of inputs, creating a large flat pool of algebraic identities, and then invokes tactics such as `ring_nf` and `linarith` in hopes of simplification. There is no effort to identify structure or extract reusable intermediate results. The tactic application is purely local and mechanical:

```
have h₂ := hf 0 0
have h₃ := hf 0 x
have h₄ := hf x 0
have h₅ := hf x (-x)
...
have h₂₆ := hf (x + x) (-x)
ring_nf at h₂ h₃ h₄ ... h₂₆ ⊢
```

In the second attempt, the prover tries to assert the final form of the solution—namely $f(x) = 2x + f(0)$—without having established why $f$ must be linear or what motivates such a guess.

It implicitly assumes the desired conclusion and attempts to work backward through aggressive simplification. This reveals a logical gap: the model never proves the Cauchy-like identity nor justifies why a linear form should even be expected.

```
have h_29 : f x = 2 * x + f 0 := by
  have h_291 := hf x 0
  ...
  ring_nf at h_291 h_292 ... ⊢
  <;> linarith
```

The third attempt generates an even larger collection of equation instances, trying all possible combinations of inputs into the original functional equation, and then offloads the burden of reasoning onto a generic decision procedure like `omega`. Again, no insight is gained; the solution depends entirely on the capacity of low-level tactics to blindly traverse the search space.

```
have h_31 : f x = 2 * x + (f 0 - 2 * 0) := by
  have h_32 := hf 0 0
  ...
  have h_42 := hf 1 (x - 1)
  ring_nf at h_32 h_33 ... h_42 ⊢
  omega
```

These degenerate strategies are a direct consequence of the reward signals guiding the training of prover models. When models are rewarded solely for producing verifiable proofs, they learn to exploit patterns that maximize verification success, not reasoning quality. Brute-force instantiation followed by tactic chains often suffices on simple benchmarks, so models internalize that behavior—even when such strategies fail to scale to Olympiad-level problems. In these more complex settings, the search space is too vast, and the necessary structural insights (such as recognizing the shifted Cauchy identity) cannot be discovered by purely local manipulations.

In contrast, our framework deliberately separates the high-level reasoning process from low-level proof verification. The Reasoner is not constrained by the demands of tactic execution or code generation; it operates at the level of abstraction and mathematical insight. By generating a chain of semantically meaningful lemmas, it defines a proof skeleton that guides the Prover and drastically reduces the search space. This separation enables the kind of reasoning that mirrors how human mathematicians approach challenging problems: by detecting invariants, proposing transformations, and narrowing the solution space through conceptual understanding. As this case study illustrates, this leads to reasoning that is not only verifiable, but also interpretable, reusable, and robust.

## C Related Work

The application of Large Language Models (LLMs) to Automated Theorem Proving has evolved rapidly. Early and some recent approaches leverage the powerful sequence modeling capabilities of LLMs to generate entire formal proofs in a single, end-to-end pass. For instance, Baldur [First et al., 2023] generates proofs for Isabelle and incorporates a repair mechanism that learns from compiler feedback to correct flawed proofs. Other works, while still operating within a largely monolithic framework, introduce internal structure. POETRY [Wang et al., 2024a] employs a recursive decomposition strategy to break down complex theorems, and LEGO-Prover [Wang et al., 2024b] hierarchically proves and reuses lemmas to manage intermediate results within its generation process. These methods treat proof generation as a sophisticated, structured sequence generation task. In contrast, our work argues that coupling high-level reasoning and low-level proof formalization within a single model limits their potential, and we instead advocate for their explicit separation.

Recognizing the limitations of direct generation, a significant line of research has focused on integrating high-level planning or sketching, mimicking human problem-solving strategies. These methods often generate a natural language plan or a structured sketch before producing the final proof. Kimina-Prover [Wang et al., 2025] achieves strong results by generating structured reasoning patterns prior to the formal proof. Similarly, DeepSeek-Prover-V2 [Ren et al., 2025], the current state-of-the-art, integrates Chain-of-Thought (CoT) style reasoning to guide its recursive subgoal decomposition pipeline. While these methods represent a conceptual step towards our approach by acknowledging the importance of planning, they still tightly couple the planning and proving phases within a single model and a fixed workflow. Our method fundamentally differs by decoupling these two stages into distinct, specialized models, allowing for more flexible and powerful interaction, such as iterative refinement of lemmas before the final proof attempt.

Our work builds upon the high-level philosophy of hierarchical proof generation, sharing conceptual similarities with prior efforts like Draft, Sketch, Prove [Jiang et al., 2023], LEGO-Prover [Wang et al., 2024b], POETRY [Wang et al., 2024a], and Subgoal-XL [Zhao et al., 2024]. The most closely related is Draft, Sketch, Prove [Jiang et al., 2023], which also employs a multi-stage pipeline: an LLM first drafts an informal proof, an autoformalizer then translates this draft into a formal sketch, and finally, an external prover completes the proof.

Despite this architectural resemblance, our approach makes a critical design choice that diverges significantly. Instead of attempting to autoformalize an entire unstructured natural language proof—a process that is itself a major research challenge and prone to semantic errors—we task our specialized Reasoner model with a more constrained and impactful objective: generating a diverse set of formal subgoal statements (lemmas). This design offers two key advantages. First, by focusing on generating strategic lemmas rather than full proof steps, we directly leverage the abstract reasoning strength of powerful LLMs to perform creative and non-trivial problem decomposition, which is essential for solving complex problems like those in the IMO competition. Second, by generating formal statements directly and leaving the proof generation to a dedicated Prover, we entirely bypass the fragile and error-prone autoformalization step. This ensures that the bridge between high-level reasoning and formal proving is both robust and precise.

# D Case Studies on IMO 2024 Problems

This section provides a detailed analysis of our framework's progress on two problems from the IMO 2024. For each problem, we present the main theorem, summarize the key sub-theorems (lemmas) that our framework successfully generated and proved, and identify the critical remaining steps required to complete the full proof.

## D.1 Analysis of IMO 2024, Problem 1

### D.1.1 Main Theorem

The problem asks to prove the equivalence between a real number $a$ being an even integer and a specific divisibility property holding for all positive integers $n$.

```
theorem imo2024_p1 (a : ℝ) :
    (∃ m : ℤ, a = 2 * m) ↔ ∀ n : ℕ, 0 < n → (n : ℤ) ∣ ∑ i in Finset.Icc 1 n, ⌊i
    ↪  * a⌋ := by
```

The proof naturally splits into two directions:

- **Forward Direction (1) $\to$ (2):** If $a = 2m$ for some integer $m$, then the divisibility property holds.
- **Reverse Direction (2) $\to$ (1):** If the divisibility property holds, then $a$ must be of the form $2m$.

### D.1.2 Progress Summary and Key Proven Lemmas

Our framework has made substantial progress on this problem, most notably by completely proving the forward direction and establishing the crucial strategic lemmas for the reverse direction.

**Forward Direction: Complete.** The framework successfully proved that if $a$ is an even integer, the divisibility property holds. This was accomplished through several lemmas, culminating in a direct proof of the implication.

```
-- Proved: The forward implication of the main theorem.
theorem imo2024_p1_forward_implication (a : ℝ) :
  (∃ m : ℤ, a = 2 * m) → (∀ n : ℕ, 0 < n → (n : ℤ) ∣ ∑ i in Finset.Icc 1 n, ⌊i *
  ↪  a⌋) := by
```

**Reverse Direction: Key Strategic Lemmas Proven.** For the more challenging reverse direction, our system proved two cornerstone lemmas that are essential to the standard human solution strategy.

1. **Periodicity of the Condition:** The framework proved that the divisibility property is periodic with a period of any even integer. This is a powerful strategic result, as it allows reducing the problem for any real number $a$ to an equivalent problem for a number in a bounded interval (e.g., $[0, 2]$).

```
-- Proved: The divisibility property is periodic by any even integer.
theorem divisibility_is_periodic_by_even_integers (a : ℝ) (m : ℤ) :
    (∀ n : ℕ, 0 < n → (n : ℤ) ∣ ∑ i in Finset.Icc 1 n, ⌊i * a⌋) ↔
    (∀ n : ℕ, 0 < n → (n : ℤ) ∣ ∑ i in Finset.Icc 1 n, ⌊i * (a - 2 *
↪  m)⌋) := by
```

2. **Special Case for Integers:** The system proved that if $a$ is an integer satisfying the divisibility property, it must be an even integer. This fully resolves the reverse direction for the specific case where $a \in \mathbb{Z}$.

```
-- Proved: An integer satisfying the property must be even.
theorem integer_must_be_even (a : ℤ)
    (h_div_int : ∀ n : ℕ, 0 < n → (n : ℤ) ∣ ∑ i in Finset.Icc 1 n, ⌊(i
↪   : ℝ) * (a : ℝ)⌋) :
    Even a := by
```

### D.1.3 Analysis of the Remaining Proof Goal

With the forward direction complete and the key periodicity lemma established, the entire proof now hinges on a single, final sub-problem.

**The Final Step: Proving the Base Case for the Periodicity.** The established lemmas allow us to reason as follows: Assume a real number $a$ satisfies the divisibility property. We can find an integer $m$ such that $a' = a - 2m$ lies in the interval $[-1, 1]$. Due to the proven periodicity, $a'$ must also satisfy the divisibility property. If we can prove that the only number in $[-1, 1]$ satisfying the property is 0, it would imply $a' = 0$, which means $a = 2m$, completing the proof.

Therefore, the critical missing lemma is to show that for any number $a \in [-1, 1]$ (or a similar interval like $(-1, 1]$), if it satisfies the property, it must be zero.

**Critical Missing Lemma**

**Goal:** To prove that if a real number $a$ in the interval $[-1, 1]$ satisfies the universal divisibility condition, then $a$ must be 0.

```
theorem univ_divisibility_in_interval_implies_zero (a : ℝ) (ha_bound : a ∈
↪  Set.Icc (-1) 1)
    (h_prop : ∀ n : ℕ, 0 < n → (n : ℤ) ∣ ∑ i in Finset.Icc 1 n, ⌊i * a⌋) :
    a = 0 := by
```

Successfully proving this final lemma would allow us to connect all the previously established results and formally complete the entire proof for IMO 2024, Problem 1. Our framework has successfully navigated the problem to its final, decisive step.

## D.2 Analysis of IMO 2024, Problem 2

### D.2.1 Main Theorem

This problem concerns a property of the greatest common divisor (GCD) of two exponential sequences. It asks to prove that the GCD becoming constant for all sufficiently large $n$ is equivalent to $a$ and $b$ both being 1.

```
theorem imo2024_p2 (a b : ℕ+) :
    (a, b) = (1, 1) ↔ ∃ g N : ℕ+, ∀ n : ℕ, N ≤ n → Nat.gcd (a^n + b) (b^n + a) =
↪  g := by
```

The proof structure involves a straightforward forward direction and a more complex reverse direction, which is typically solved by considering cases.

### D.2.2 Progress Summary and Key Proven Lemmas

Our framework successfully proved the simple forward direction and made significant headway on the reverse direction by proving the special case where $a = b$.

**Forward Direction: Complete.** The framework easily proved that if $a = 1$ and $b = 1$, the GCD sequence is constant. In this case, $\gcd(1^n + 1, 1^n + 1) = \gcd(2, 2) = 2$ for all $n$, so one can choose $g = 2$ and $N = 1$.

```
-- Proved: The forward implication of the main theorem.
theorem imo2024_p2_forward_implication (a b : ℕ+) :
  (a, b) = (1, 1) → ∃ g N : ℕ+, ∀ n : ℕ, N ≤ n → Nat.gcd (a^n + b) (b^n + a) = g
  ↪  := by
```

**Reverse Direction: Special Case $a = b$ Proven.** For the reverse direction, the framework identified and fully proved the crucial sub-case where $a = b$. It correctly deduced that if $a = b$ and the GCD property holds, then $a$ must be 1. The reasoning relies on the fact that if $a = b$, the GCD is $\gcd(a^n + a, a^n + a) = a^n + a$. For this sequence to be constant for $n \geq N$, $a$ cannot be greater than 1.

```
-- Proved: If a=b and the GCD property holds, then a=1 and b=1.
theorem imo2024_p2_bwd_a_eq_b (a b : ℕ+) (h_ab : a = b)
    (h_gcd_const : ∃ g N : ℕ+, ∀ n : ℕ, N ≤ n → Nat.gcd (a^n + b) (b^n + a) = g)
    ↪  :
    (a, b) = (1, 1) := by
```

This lemma is supported by another proven sub-theorem stating that an exponential sequence like $a^n + a$ cannot be eventually constant if $a > 1$.

```
-- Proved: An exponential sequence is not eventually constant for a > 1.
theorem exponential_not_eventually_constant (a : ℕ+) :
  a > 1 → ¬∃ g N : ℕ+, ∀ n : ℕ, N ≤ n → a^n + a = g := by
```

### D.2.3 Analysis of the Remaining Proof Goal

With the forward direction and the $a = b$ case of the reverse direction complete, the entire proof now rests on resolving the case where $a \neq b$.

**The Final Step: Proving the Case $a \neq b$ Leads to a Contradiction.** The standard human approach for the case $a \neq b$ (without loss of generality, assume $a > b$) is to show that the GCD sequence, $d_n = \gcd(a^n + b, b^n + a)$, cannot be eventually constant if $a > 1$. A common technique involves using properties of the GCD, such as $\gcd(X, Y) = \gcd(X, Y - kX)$. Applying this here:

$$d_n = \gcd(a^n + b, b^n + a) = \gcd(a^n + b, b^n + a - b^{n-1}(a^n + b))$$

which simplifies the second term. The key is to show that if $a > b \geq 1$, this sequence cannot be constant for large $n$.

Therefore, the critical missing lemma is to prove by contradiction that if the GCD property holds, the case $a \neq b$ is impossible unless $a = b = 1$ (which is already covered).

Critical Missing Lemma

**Goal:** To prove that if $a \neq b$, the GCD property cannot hold. A common way is to show that if $a > b$, the GCD sequence is not constant.

```
-- This lemma is stated to lead to a contradiction with the main hypothesis.
theorem gcd_is_not_eventually_constant_if_unequal (a b : ℕ+) (h_neq : a ≠ b)
↪  :
    ¬(∃ g N : ℕ+, ∀ n : ℕ, N ≤ n → Nat.gcd (a^n + b) (b^n + a) = g) := by
-- A more direct approach to prove is:
-- if a > b >= 1, then the sequence is not constant.
```

Or, framing it to directly complete the main proof:

```
-- This lemma, combined with the a=b case, would complete the proof.
theorem p2_bwd_dir_a_neq_b (a b : ℕ+) :
  (∃ g N : ℕ+, ∀ n : ℕ, N ≤ n → Nat.gcd (a^n + b) (b^n + a) = g) → a ≠ b
  ↪  → False := by
```

By proving that the GCD property cannot hold for distinct positive integers $a$ and $b$, our framework would successfully eliminate the only remaining case, thereby completing the proof for IMO 2024, Problem 2.

## E Solved IMO problems

### E.1 IMO 2020 P2

```
-- Solution to IMO 2020 P2 by DRP-IMO

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/--Consider four real numbers \( a, b, c, \) and \( d \) such that \( 0 < d \leq c
↪  \leq b \leq a \) and their sum is equal to 1, i.e., \( a + b + c + d = 1 \).
↪  Prove that the product of the weighted sum \( a + 2b + 3c + 4d \) and the sum of
↪  their squares \( a^2 + b^2 + c^2 + d^2 \) is less than 1.-/
theorem weighted_sum_times_sum_sq_lt_one (a b c d : ℝ) (hd_pos : 0 < d) (hdc : d ≤
↪  c) (hcb : c ≤ b) (hba : b ≤ a) (h_sum_eq_1 : a + b + c + d = 1) :
  (a + 2*b + 3*c + 4*d) * (a^2 + b^2 + c^2 + d^2) < 1 := by
  have h_a_lt_one : a < 1 := by
    have h₁ : a + b + c + d = 1 := h_sum_eq_1
    have h₂ : 0 < b := by linarith
    have h₃ : 0 < c := by linarith
    have h₄ : 0 < d := hd_pos
    nlinarith

  have h_b_lt_one : b < 1 := by
    have h₁ : a ≥ b := by linarith
    have h₂ : a < 1 := h_a_lt_one
    nlinarith

  have h_c_lt_one : c < 1 := by
    have h₁ : b ≥ c := by linarith
    have h₂ : b < 1 := h_b_lt_one
    nlinarith

  have h_d_lt_one : d < 1 := by
    have h₁ : c ≥ d := by linarith
    have h₂ : c < 1 := h_c_lt_one
```

```
    nlinarith

  have h_a2_lt_a : a^2 < a := by
    have h₁ : a < 1 := h_a_lt_one
    have h₂ : a > 0 := by linarith
    nlinarith [sq_pos_of_pos h₂]
    <;> nlinarith

  have h_b2_lt_b : b^2 < b := by
    have h₁ : b < 1 := h_b_lt_one
    have h₂ : b > 0 := by linarith
    nlinarith [sq_pos_of_pos h₂]
    <;> nlinarith

  have h_c2_lt_c : c^2 < c := by
    have h₁ : c < 1 := h_c_lt_one
    have h₂ : c > 0 := by linarith
    nlinarith [sq_pos_of_pos h₂]
    <;> nlinarith

  have h_d2_lt_d : d^2 < d := by
    have h₁ : d < 1 := h_d_lt_one
    have h₂ : d > 0 := hd_pos
    nlinarith [sq_pos_of_pos h₂]
    <;> nlinarith

  have h_sum_sq_lt_one : a^2 + b^2 + c^2 + d^2 < 1 := by
    nlinarith [h_a2_lt_a, h_b2_lt_b, h_c2_lt_c, h_d2_lt_d]
    <;> linarith

  have h_main : (a + 2*b + 3*c + 4*d) * (a^2 + b^2 + c^2 + d^2) < 1 := by
    have h₁ : 0 < a + 2 * b + 3 * c + 4 * d := by
      nlinarith [hd_pos, hcb, hba, hdc, h_sum_eq_1]
    have h₂ : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 < 1 := h_sum_sq_lt_one
    nlinarith [h₁, h₂]
    <;> nlinarith

  exact h_main


theorem vars_are_in_0_1 (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b)
↪  (hba : b ≤ a) (h1 : a + b + c + d = 1) :
  (0 < a ∧ a < 1) ∧ (0 < b ∧ b < 1) ∧ (0 < c ∧ c < 1) ∧ (0 < d ∧ d < 1) := by
  have h_a_pos : 0 < a := by
    nlinarith [hdc, hcb, hba, hd0, h1]
    <;> nlinarith

  have h_a_lt_1 : a < 1 := by
    have h2 : a < 1 := by
      nlinarith [h1, h_a_pos, hba, hcb, hdc, hd0]
    exact h2

  have h_b_pos : 0 < b := by
    nlinarith [hdc, hcb, hba, hd0, h1]

  have h_b_lt_1 : b < 1 := by
    have h2 : b < 1 := by
      nlinarith [h1, h_a_pos, h_a_lt_1, hba, hcb, hdc, hd0]
    exact h2

  have h_c_pos : 0 < c := by
    nlinarith [hdc, hcb, hba, hd0, h1]
```

```

  have h_c_lt_1 : c < 1 := by
    have h2 : c < 1 := by
      nlinarith [h1, h_a_pos, h_a_lt_1, h_b_pos, h_b_lt_1, hba, hcb, hdc, hd0]
    exact h2

  have h_d_pos : 0 < d := by
    exact hd0

  have h_d_lt_1 : d < 1 := by
    have h2 : d < 1 := by
      nlinarith [h1, h_a_pos, h_a_lt_1, h_b_pos, h_b_lt_1, h_c_pos, h_c_lt_1, hdc, hcb, hba, hd0]
    exact h2

  refine' ⟨⟨h_a_pos, h_a_lt_1⟩, ⟨h_b_pos, h_b_lt_1⟩, ⟨h_c_pos, h_c_lt_1⟩, ⟨h_d_pos, h_d_lt_1⟩⟩


theorem imo2020_q2 (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a) (h1 : a + b + c + d = 1) :
    (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) < 1 := by
  -- strategy:
  -- 1. apply weighted AM-GM inequality to prove a^a * b^b * c^c * d^d ≤ a^2 + b^2 + c^2 + d^2
  -- 2. ues subgoal 'weighted_sum_times_sum_sq_lt_one' to get (a + 2*b + ...) * (a^2 + b^2 + ...) < 1
  -- 3. combine both results to reach final conclusion

  -- define S
  let S := a^2 + b^2 + c^2 + d^2

  -- step 1: apply weighted AM-GM inequality
  -- we need to prove a^a * b^b * c^c * d^d ≤ S
  have h_geom_mean_le_sum_sq : a ^ a * b ^ b * c ^ c * d ^ d ≤ S := by
    -- in order to use the subgoal 'geom_mean_le_arith_mean_weighted', we use Fin 4 as an index type
    let w : Fin 4 → ℝ := ![a, b, c, d]
    let z : Fin 4 → ℝ := ![a, b, c, d]

    -- check AM-GM prerequisite
    have h_pos_conds : (0 < a) ∧ (0 < b) ∧ (0 < c) ∧ (0 < d) := by
      have h_all := vars_are_in_0_1 a b c d hd0 hdc hcb hba h1
      exact ⟨h_all.1.1, h_all.2.1.1, h_all.2.2.1.1, h_all.2.2.2.1⟩

    -- 1. non-negative weights
    have h_weights_nonneg : ∀ i, 0 ≤ w i := by
      intro i; fin_cases i <;> simp [w] <;> linarith [h_pos_conds.1, h_pos_conds.2.1, h_pos_conds.2.2.1, h_pos_conds.2.2.2]

    -- 2. weights sum-up to 1
    have h_weights_sum_1 : ∑ i, w i = 1 := by
      simp [w, Fin.sum_univ_four, h1]

    -- 3. non-negative values
    have h_values_nonneg : ∀ i, 0 ≤ z i := by
      intro i; fin_cases i <;> simp [z] <;> linarith [h_pos_conds.1, h_pos_conds.2.1, h_pos_conds.2.2.1, h_pos_conds.2.2.2]

    -- use the subgoal based on AM-GM
    have h_am_gm := geom_mean_le_arith_mean_weighted (Finset.univ) w z (fun i _ ↦ h_weights_nonneg i) h_weights_sum_1 (fun i _ ↦ h_values_nonneg i)

```

```
    -- transform AM-GM results to the form we want
    -- `simp` will handle a*a -> a^2
    simp only [Fin.prod_univ_four, Fin.sum_univ_four, w, z, ← pow_two] at h_am_gm

    -- it will replace 'S' to 'a^2 + b^2 + c^2 + d^2'
    unfold S
    -- now the target fully matchs 'h_am_gm'
    exact h_am_gm

  -- step 2: get results from key lemmas
  have h_main_ineq : (a + 2 * b + 3 * c + 4 * d) * S < 1 := by
    exact weighted_sum_times_sum_sq_lt_one a b c d hd0 hdc hcb hba h1

  -- step 3 & 4 & 5: assumble final proof
  calc
    (a + 2*b + 3*c + 4*d) * (a^a * b^b * c^c * d^d)
    -- first, use the results from step 1, we need to prove (a + 2*b + ...) is positive
    -- lemma 'vars_are_in_0_1' guarantees a,b,c,d > 0, thus their weighted sum also > 0
    _ ≤ (a + 2*b + 3*c + 4*d) * S := by
        apply mul_le_mul_of_nonneg_left h_geom_mean_le_sum_sq
        have h_pos_conds := vars_are_in_0_1 a b c d hd0 hdc hcb hba h1
        linarith [h_pos_conds.1.1, h_pos_conds.2.1.1, h_pos_conds.2.2.1.1, h_pos_conds.2.2.2.1]
    -- then, use the results from step 2 to finish proving
    _ < 1 := h_main_ineq
```

### E.2 IMO 2019 P1

```
-- Solution to IMO 2019 P1 by DRP-IMO

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

def solution_set (f : ℤ → ℤ) : Prop :=
  (∀ x : ℤ, f x = 0) ∨ ∃ c : ℤ, ∀ x : ℤ, f x = 2 * x + c

theorem linear_form_plus_f_f_x_implies_solutions (f : ℤ → ℤ) (c : ℤ)
    (h_f_form : ∀ x, f x = c * x + f 0) (h_f_f_x : ∀ x, f (f x) = 2 * f x + f 0) :
  (∀ x, f x = 0) ∨ (∀ x, f x = 2 * x + f 0) := by
  have h_c_squared : c^2 = 2 * c := by
    have h1 := h_f_f_x 1
    have h2 := h_f_f_x 0
    have h3 := h_f_f_x (-1)
    have h4 := h_f_form 1
    have h5 := h_f_form 0
    have h6 := h_f_form (-1)
    have h7 := h_f_form (f 1)
    have h8 := h_f_form (f 0)
    have h9 := h_f_form (f (-1))
    have h10 := h_f_f_x (f 1)
    have h11 := h_f_f_x (f 0)
    have h12 := h_f_f_x (f (-1))
    have h13 := h_f_form (c * 1 + f 0)
    have h14 := h_f_form (c * 0 + f 0)
    have h15 := h_f_form (c * (-1) + f 0)
    have h16 := h_f_form (c * (f 1) + f 0)
```

```
    have h17 := h_f_form (c * (f 0) + f 0)
    have h18 := h_f_form (c * (f (-1)) + f 0)
    ring_nf at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 ⊢
    nlinarith [sq_nonneg (c - 2), sq_nonneg (c + 2), sq_nonneg (c - 1), sq_nonneg (c + 1)]

  have h_c_cases : c = 0 ∨ c = 2 := by
    have h₁ : c ^ 2 = 2 * c := h_c_squared
    have h₂ : c = 0 ∨ c = 2 := by
      have h₃ : c * (c - 2) = 0 := by
        linarith
      have h₄ : c = 0 ∨ c - 2 = 0 := by
        apply eq_zero_or_eq_zero_of_mul_eq_zero h₃
      cases h₄ with
      | inl h₄ =>
        exact Or.inl h₄
      | inr h₄ =>
        have h₅ : c = 2 := by
          omega
        exact Or.inr h₅
    exact h₂

  have h_main : (∀ x, f x = 0) ∨ (∀ x, f x = 2 * x + f 0) := by
    cases h_c_cases with
    | inl h_c_zero =>
      -- Case c = 0
      have h_f_zero : ∀ x, f x = f 0 := by
        intro x
        have h₁ := h_f_form x
        simp [h_c_zero] at h₁ ⊢
        <;> linarith
      have h_f_zero_zero : f 0 = 0 := by
        have h₁ := h_f_f_x 0
        have h₂ := h_f_form 0
        have h₃ := h_f_form (f 0)
        have h₄ := h_f_f_x (f 0)
        simp [h_f_zero] at h₁ h₂ h₃ h₄ ⊢
        <;>
        (try omega) <;>
        (try
          {
            nlinarith [h_f_form 0, h_f_form 1, h_f_form (-1), h_f_form (f 0)]
          }) <;>
        (try
          {
            cases' h_c_cases with h_c_zero h_c_two <;> simp_all [h_c_zero, h_c_two] <;>
            (try omega) <;>
            (try nlinarith) <;>
            (try linarith)
          }) <;>
        (try
          {
            aesop
          })
      have h_f_zero_all : ∀ x, f x = 0 := by
        intro x
        have h₁ := h_f_zero x
        have h₂ := h_f_zero 0
        have h₃ := h_f_zero (-1)
        have h₄ := h_f_zero 1
        simp [h_f_zero_zero] at h₁ h₂ h₃ h₄ ⊢
```

```
        <;>
        (try omega) <;>
        (try nlinarith) <;>
        (try aesop)
        <;>
        (try
          {
            simp_all [h_f_form, h_c_zero]
            <;>
            (try omega) <;>
            (try nlinarith) <;>
            (try aesop)
          })
      exact Or.inl h_f_zero_all
    | inr h_c_two =>
      -- Case c = 2
      have h_f_form_two : ∀ x, f x = 2 * x + f 0 := by
        intro x
        have h₁ := h_f_form x
        simp [h_c_two] at h₁ ⊢
        <;> linarith
      exact Or.inr h_f_form_two

  exact h_main

theorem prop_cauchy_like (f : ℤ → ℤ) (h_f_all : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b)))
    (h_f_f_x : ∀ x, f (f x) = 2 * f x + f 0) (h_f_2x : ∀ x, f (2 * x) = 2 * f x - f 0) (x y : ℤ) :
  f (x + y) = f x + f y - f 0 := by
  have h_main : f (x + y) = f x + f y - f 0 := by
    have h1 := h_f_all (x + y) 0
    have h2 := h_f_all x y
    have h3 := h_f_all (x + y) y
    have h4 := h_f_all x (x + y)
    have h5 := h_f_2x (x + y)
    have h6 := h_f_2x x
    have h7 := h_f_2x y
    have h8 := h_f_all 0 (x + y)
    have h9 := h_f_all 0 x
    have h10 := h_f_all 0 y
    have h11 := h_f_f_x (x + y)
    have h12 := h_f_f_x x
    have h13 := h_f_f_x y
    have h14 := h_f_all (2 * (x + y)) 0
    have h15 := h_f_all (2 * x) 0
    have h16 := h_f_all (2 * y) 0
    have h17 := h_f_all x 0
    have h18 := h_f_all y 0
    have h19 := h_f_all (x + y) (x + y)
    have h20 := h_f_all x x
    have h21 := h_f_all y y
    -- Simplify the expressions using the given conditions
    simp [h_f_2x, mul_add, add_mul, mul_comm, mul_left_comm, mul_assoc] at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 ⊢
    <;> ring_nf at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 ⊢
    <;> omega
  exact h_main

theorem prop_f_f_x (f : ℤ → ℤ) (h_f_all : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b))) (x : ℤ) :
  f (f x) = 2 * f x + f 0 := by
```

```
  have h_main : f (f x) = 2 * f x + f 0 := by
    have h1 := h_f_all x 0
    have h2 := h_f_all 0 x
    have h3 := h_f_all x x
    have h4 := h_f_all (-x) x
    have h5 := h_f_all x (-x)
    have h6 := h_f_all 0 0
    have h7 := h_f_all x (-2 * x)
    have h8 := h_f_all (-x) (-x)
    have h9 := h_f_all x 1
    have h10 := h_f_all x (-1)
    have h11 := h_f_all 1 x
    have h12 := h_f_all (-1) x
    have h13 := h_f_all 1 0
    have h14 := h_f_all (-1) 0
    have h15 := h_f_all 0 1
    have h16 := h_f_all 0 (-1)
    have h17 := h_f_all 1 1
    have h18 := h_f_all (-1) (-1)
    -- Simplify the equations to find a relationship between f(0) and f(f(0))
    simp at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
    ring_nf at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 ⊢
    -- Use linear arithmetic to solve for the desired result
    omega
  exact h_main

theorem prop_f_2x (f : ℤ → ℤ) (h_f_all : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a +
↪  b)))
    (h_f_f_x : ∀ x, f (f x) = 2 * f x + f 0) (x : ℤ) :
  f (2 * x) = 2 * f x - f 0 := by
  have h1 : f (f (2 * x)) = 2 * f (2 * x) + f 0 := by
    have h1 := h_f_f_x (2 * x)
    -- Simplify the expression using the given condition h_f_f_x
    simp at h1 ⊢
    <;> linarith

  have h2 : f (2 * x) + 2 * f x = f (f (2 * x)) := by
    have h2 := h_f_all x x
    -- Simplify the expression using the given condition h_f_all
    ring_nf at h2 ⊢
    <;> linarith

  have h3 : f (2 * x) + 2 * f x = 2 * f (2 * x) + f 0 := by
    have h3 : f (2 * x) + 2 * f x = f (f (2 * x)) := h2
    rw [h3]
    have h4 : f (f (2 * x)) = 2 * f (2 * x) + f 0 := h1
    rw [h4]
    <;> ring
    <;> omega

  have h4 : f (2 * x) = 2 * f x - f 0 := by
    have h5 : f (2 * x) + 2 * f x = 2 * f (2 * x) + f 0 := h3
    -- Rearrange the equation to isolate f(2 * x)
    have h6 : f (2 * x) = 2 * f x - f 0 := by
      -- Solve for f(2 * x) using linear arithmetic
      linarith
    exact h6

  apply h4


theorem cauchy_implies_linear_form (f : ℤ → ℤ) (h_cauchy_like : ∀ x y, f (x + y) =
↪  f x + f y - f 0) :
```

```
  ∃ c : ℤ, ∀ x, f x = c * x + f 0 := by
  have h_main : ∃ (c : ℤ), ∀ (x : ℤ), f x = c * x + f 0 := by
    use f 1 - f 0
    intro x
    have h1 : ∀ n : ℤ, f n = (f 1 - f 0) * n + f 0 := by
      intro n
      induction n using Int.induction_on with
      | hz =>
        -- Base case: n = 0
        simp [h_cauchy_like]
        <;> ring_nf
        <;> omega
      | hp n ih =>
        -- Inductive step: n = p + 1
        have h2 := h_cauchy_like n 1
        have h3 := h_cauchy_like 0 (n + 1)
        have h4 := h_cauchy_like (n + 1) 0
        have h5 := h_cauchy_like 1 0
        have h6 := h_cauchy_like 0 1
        simp at h2 h3 h4 h5 h6
        simp [ih, add_mul, mul_add, mul_one, mul_neg, mul_zero, sub_eq_add_neg] at
        ↪  h2 h3 h4 h5 h6 ⊢
        <;> ring_nf at *
        <;> omega
      | hn n ih =>
        -- Inductive step: n = - (n + 1)
        have h2 := h_cauchy_like (-n - 1) 1
        have h3 := h_cauchy_like 0 (-n - 1)
        have h4 := h_cauchy_like (-n - 1) 0
        have h5 := h_cauchy_like 1 0
        have h6 := h_cauchy_like 0 1
        simp at h2 h3 h4 h5 h6
        simp [ih, add_mul, mul_add, mul_one, mul_neg, mul_zero, sub_eq_add_neg] at
        ↪  h2 h3 h4 h5 h6 ⊢
        <;> ring_nf at *
        <;> omega
    have h2 := h1 x
    have h3 := h1 1
    have h4 := h1 0
    simp at h2 h3 h4 ⊢
    <;> linarith
  exact h_main

theorem step6_zero_function_is_solution (f : ℤ → ℤ) (h_zero : ∀ x, f x = 0) : (∀ a
↪  b, f (2 * a) + 2 * (f b) = f (f (a + b))) := by
have h_main : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b)) := by
  intro a b
  have h1 : f (2 * a) = 0 := by
    rw [h_zero]
    <;> simp [h_zero]
  have h2 : f b = 0 := by
    rw [h_zero]
    <;> simp [h_zero]
  have h3 : f (a + b) = 0 := by
    rw [h_zero]
    <;> simp [h_zero]
  have h4 : f (f (a + b)) = 0 := by
    rw [h_zero]
    <;> simp [h_zero]
  -- Simplify the LHS and RHS using the above equalities
  simp [h1, h2, h3, h4, h_zero]
  <;> linarith
```

```
exact h_main

theorem step7_linear_function_is_solution (f : ℤ → ℤ) (c : ℤ) (h_lin : ∀ x, f x =
↪  2 * x + c) : (∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b))) := by
have h_main : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b)) := by
  intro a b
  have h1 : f (2 * a) = 2 * (2 * a) + c := by
    rw [h_lin]
    <;> ring
  have h2 : f b = 2 * b + c := by
    rw [h_lin]
    <;> ring
  have h3 : f (f (a + b)) = f (2 * (a + b) + c) := by
    have h4 : f (a + b) = 2 * (a + b) + c := by
      rw [h_lin]
      <;> ring
    rw [h4]
    <;> ring
  have h4 : f (f (a + b)) = 2 * (2 * (a + b) + c) + c := by
    rw [h3]
    rw [h_lin]
    <;> ring
  have h5 : f (2 * a) + 2 * (f b) = (2 * (2 * a) + c) + 2 * (2 * b + c) := by
    rw [h1, h2]
    <;> ring
  have h6 : f (2 * a) + 2 * (f b) = 4 * a + 4 * b + 3 * c := by
    linarith
  have h7 : f (f (a + b)) = 4 * a + 4 * b + 3 * c := by
    linarith
  linarith
exact h_main

theorem imo2019_p1
    (f : ℤ → ℤ) :
    (∀ a b : ℤ, f (2 * a) + 2 * f b = f (f (a + b))) ↔ solution_set f := by
  constructor
  · intro h_fe
    have h_ff : ∀ x, f (f x) = 2 * f x + f 0 :=
      prop_f_f_x f h_fe
    have h_f2 : ∀ x, f (2 * x) = 2 * f x - f 0 :=
      prop_f_2x f h_fe h_ff
    have h_add : ∀ x y, f (x + y) = f x + f y - f 0 :=
      prop_cauchy_like f h_fe h_ff h_f2
    rcases cauchy_implies_linear_form f h_add with ⟨c, h_lin0⟩
    have h_split :
        (∀ x, f x = 0) ∨ (∀ x, f x = 2 * x + f 0) :=
      linear_form_plus_f_f_x_implies_solutions f c h_lin0 h_ff
    cases h_split with
    | inl h0 =>
        exact Or.inl h0
    | inr h2 =>
        exact Or.inr ⟨f 0, h2⟩
  · intro h_sol
    cases h_sol with
    | inl h0 =>
        exact step6_zero_function_is_solution f h0
    | inr h_exists =>
        rcases h_exists with ⟨c, h_lin⟩
        exact step7_linear_function_is_solution f c h_lin
```

### E.3 IMO 2011 P3

```
-- Solution to IMO 2011 P3 by DRP-IMO

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat



theorem imo2011_p3_lemma1_f_neg_le_self (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x)) :
  ∀ x, f x < 0 → f x ≤ x := by
  have h_main : ∀ (x : ℝ), f x < 0 → f x ≤ x := by
    intro x hx
    have h1 : f x ^ 2 - x * f x ≥ 0 := by
      have h2 := hf x (f x - x)
      have h3 := hf (f x) (x - f x)
      have h4 := hf x 0
      have h5 := hf 0 x
      have h6 := hf x x
      have h7 := hf x (-x)
      have h8 := hf (-x) x
      have h9 := hf 0 0
      have h10 := hf x 1
      have h11 := hf 1 x
      have h12 := hf x (-1)
      have h13 := hf (-1) x
      have h14 := hf x (f x)
      have h15 := hf (f x) x
      have h16 := hf x (-f x)
      have h17 := hf (-f x) x
      have h18 := hf x (x + f x)
      have h19 := hf (x + f x) x
      have h20 := hf x (-x)
      have h21 := hf (-x) x
      have h22 := hf x (x - f x)
      have h23 := hf (x - f x) x
      have h24 := hf x (f x + x)
      have h25 := hf (f x + x) x
      have h26 := hf x (2 * f x)
      have h27 := hf (2 * f x) x
      have h28 := hf x (-2 * f x)
      have h29 := hf (-2 * f x) x
      -- Normalize the expressions to simplify the inequalities
      ring_nf at h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21 h22 h23 h24 h25 h26 h27 h28 h29 ⊢
      -- Use linear arithmetic to prove the inequality
      nlinarith [sq_nonneg (f x - x), sq_nonneg (f x + x), sq_nonneg (f x - 2 * x), sq_nonneg (f x + 2 * x),
        sq_nonneg (2 * f x - x), sq_nonneg (2 * f x + x)]
    have h3 : f x ≤ x := by
      by_contra h
      have h4 : f x > x := by linarith
      have h5 : f x ^ 2 - x * f x < 0 := by
        nlinarith [hx, h4]
      nlinarith
    exact h3
  exact h_main
```

```lean
/--Let \( f : \mathbb{R} \to \mathbb{R} \) be a function such that for all real numbers \( x \) and \( y \), the inequality \( f(x + y) \leq y \cdot f(
x) + f(f(x)) \) holds. Prove that for all real numbers \( x \), the inequality \( f(x) \leq f(f(x)) \) is true.-/
theorem imo2011_p3_st1 (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x)) :
  ∀ x, f x ≤ f (f x) := by
  have h_main : ∀ (x : ℝ), f x ≤ f (f x) := by
    intro x
    have h₁ := hf x 0
    -- Simplify the inequality by substituting y = 0
    simp at h₁
    -- Use the simplified inequality to conclude the proof
    linarith
  exact h_main



/--Consider a function \( f : \mathbb{R} \to \mathbb{R} \) that satisfies the condition: for all real numbers \( x \) and \( y \), \( f(x + y) \leq y \
cdot f(x) + f(f(x)) \). Prove that for all real numbers \( x \), \( f(x) \leq 0 \).-/
theorem aux_f_nonpositive (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x)) : ∀ x, f x ≤ 0 := by
have h_main : ∀ x, f x ≤ 0 := by
  intro x
  by_contra h
  have h₁ : f x > 0 := by linarith
  have h₂ := hf x (-x)
  have h₃ := hf 0 (f x)
  have h₄ := hf x 0
  have h₅ := hf (-x) x
  have h₆ := hf (-x) (-x)
  have h₇ := hf x (f x)
  have h₈ := hf (f x) (-f x)
  have h₉ := hf (f x) 0
  have h₁₀ := hf 0 (-f x)
  have h₁₁ := hf x (2 * x)
  have h₁₂ := hf x (-2 * x)
  have h₁₃ := hf (2 * x) (-x)
  have h₁₄ := hf (2 * x) x
  have h₁₅ := hf (-2 * x) x
  have h₁₆ := hf (-2 * x) (-x)
  have h₁₇ := hf (f x) x
  have h₁₈ := hf (f x) (-x)
  have h₁₉ := hf x (f x)
  have h₂₀ := hf (-x) (f x)
  have h₂₁ := hf x (-f x)
  have h₂₂ := hf (-x) (-f x)
  have h₂₃ := hf (2 * f x) (-f x)
  have h₂₄ := hf (-2 * f x) (-f x)
  have h₂₅ := hf (2 * f x) (f x)
  have h₂₆ := hf (-2 * f x) (f x)
  have h₂₇ := hf (f x) (2 * f x)
  have h₂₈ := hf (f x) (-2 * f x)
  have h₂₉ := hf x (x)
  have h₃₀ := hf x (-x)
  have h₃₁ := hf 0 (2 * f x)
  have h₃₂ := hf 0 (-2 * f x)
  have h₃₃ := hf (2 * f x) 0
  have h₃₄ := hf (-2 * f x) 0
  have h₃₅ := hf (f x) (f x)
  have h₃₆ := hf (-f x) (f x)
  have h₃₇ := hf (f x) (-f x)
```

```
  have h_38 := hf (-f x) (-f x)
  norm_num at *
  <;>
  (try nlinarith) <;>
  (try linarith) <;>
  (try nlinarith [h_1, h_2, h_3, h_4, h_5, h_6, h_7, h_8, h_9, h_10, h_11, h_12, h_13, h_14, h_15, h_16, h_17, h_18, h_19, h_20, h_21, h_22, h_23, h_24, h_25, h_26, h_27, h_28, h_2
9, h_30, h_31, h_32, h_33, h_34, h_35, h_36, h_37, h_38]) <;>
  (try
    nlinarith [hf 0 0, hf x 0, hf 0 x, hf x (-x), hf (-x) x, hf (x + x) (-x), hf (-x) (x + x), hf (x - x) (x + x), hf (x + x) (x - x)]) <;>
  (try
    nlinarith [hf 0 0, hf x 0, hf 0 x, hf x (-x), hf (-x) x, hf (x + x) (-x), hf (-x) (x + x), hf (x - x) (x + x), hf (x + x) (x - x)]) <;>
  (try
    nlinarith [hf 0 0, hf x 0, hf 0 x, hf x (-x), hf (-x) x, hf (x + x) (-x), hf (-x) (x + x), hf (x - x) (x + x), hf (x + x) (x - x)])
  <;>
  nlinarith

exact h_main



theorem lemma_final_implication (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x))
  (h_f_at_0_is_0 : f 0 = 0) (h_f_non_positive : ∀ x, f x ≤ 0) : ∀ x ≤ 0, f x = 0 := by
  have h_main : ∀ (x : ℝ), x ≤ 0 → f x = 0 := by
    intro x hx
    have h1 : f x = 0 := by
      by_cases hx0 : x = 0
      · -- If x = 0, then f(0) = 0 by hypothesis
        simp [hx0, h_f_at_0_is_0]
      · -- If x ≠ 0, then x < 0
        have hx1 : x < 0 := by
          cases' lt_or_gt_of_ne hx0 with h h
          · linarith
          · exfalso
            linarith
        -- Use the given inequality with y = x and y = -x to derive the desired result
        have h2 := hf x x
        have h3 := hf (-x) x
        have h4 := hf x (-x)
        have h5 := hf 0 x
        have h6 := hf x 0
        have h7 := hf 0 (-x)
        have h8 := hf (-x) 0
        -- Simplify the inequalities using the given conditions
        norm_num [h_f_at_0_is_0] at h2 h3 h4 h5 h6 h7 h8 ⊢
        nlinarith [h_f_non_positive x, h_f_non_positive (-x), h_f_non_positive (f x),
          h_f_non_positive (x + x), h_f_non_positive (x - x), h_f_non_positive 0]
    exact h1
  exact h_main

/--Let \( f : \mathbb{R} \to \mathbb{R} \) be a function satisfying the inequality \( f(x + y) \leq y \cdot f(x) + f(f(x)) \) for all real numbers \( x
 \) and \( y \). Suppose that \( f(0) = c \) and there exists some \( x_0 \) such that \( f(x_0) = 0 \). Prove that \( c \geq 0 \), \( f(c) = c \), \(
f(y) \leq c \) for all real numbers \( y \), and \( f(y) \leq c \cdot y + c \) for all real numbers \( y \).-/
```

```
theorem lemma_properties_if_f_has_zero (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x))
  (h_f0_eq_c : f 0 = c) (hx₀ : ∃ x₀, f x₀ = 0) :
  c ≥ 0 ∧ f c = c ∧ (∀ y, f y ≤ c) ∧ (∀ y, f y ≤ c * y + c) := by
  have h_c_ge_zero : c ≥ 0 := by
    obtain ⟨x₀, hx₀⟩ := hx₀
    have h1 := hf x₀ (-x₀)
    have h2 := hf 0 (-x₀)
    have h3 := hf x₀ 0
    have h4 := hf 0 0
    have h5 := hf x₀ (f x₀)
    have h6 := hf 0 (f 0)
    have h7 := hf x₀ (-f x₀)
    have h8 := hf 0 (-f 0)
    norm_num [h_f0_eq_c, hx₀] at *
    <;>
    (try linarith) <;>
    (try nlinarith) <;>
    (try simp_all [h_f0_eq_c, hx₀]) <;>
    (try linarith) <;>
    (try nlinarith)
    <;>
    (try
      nlinarith [sq_nonneg (f x₀), sq_nonneg (f 0), sq_nonneg (x₀ + 0), sq_nonneg (x₀ - 0), sq_nonneg (f x₀ + f 0), sq_nonneg (f x₀ - f 0)])
    <;>
    (try
      nlinarith [sq_nonneg (f x₀), sq_nonneg (f 0), sq_nonneg (x₀ + 0), sq_nonneg (x₀ - 0), sq_nonneg (f x₀ + f 0), sq_nonneg (f x₀ - f 0)])

  have h_f_c_eq_c : f c = c := by
    obtain ⟨x₀, hx₀⟩ := hx₀
    have h₁ := hf x₀ (c - x₀)
    have h₂ := hf 0 (c)
    have h₃ := hf c (-c)
    have h₄ := hf x₀ 0
    have h₅ := hf 0 0
    have h₆ := hf x₀ (f x₀)
    have h₇ := hf 0 (f 0)
    have h₈ := hf x₀ (-x₀)
    have h₉ := hf 0 (-x₀)
    simp [h_f0_eq_c, hx₀] at h₁ h₂ h₃ h₄ h₅ h₆ h₇ h₈ h₉ ⊢
    <;>
    (try ring_nf at * <;> nlinarith) <;>
    (try
      {
        nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀)]
      }) <;>
    (try
      {
        nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀), sq_nonneg (f c)]
      })
    <;>
    (try
      {
        nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀), sq_nonneg (f c), sq_nonneg (f 0)]
      })
    <;>
    (try
      {
```

```
        nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀),
        ↪  sq_nonneg (f c), sq_nonneg (f 0), sq_nonneg (c - f x₀)]
      })
    <;>
    (try
      {
        nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀),
        ↪  sq_nonneg (f c), sq_nonneg (f 0), sq_nonneg (c - f x₀), sq_nonneg (f x₀
        ↪  -
c)]
      })

  have h_f_le_c : ∀ y, f y ≤ c := by
    intro y
    have h1 := hf y (-y)
    have h2 := hf y (c - y)
    have h3 := hf 0 (y)
    have h4 := hf c (-c)
    have h5 := hf y 0
    have h6 := hf 0 0
    have h7 := hf c 0
    have h8 := hf 0 c
    have h9 := hf y (f y)
    have h10 := hf 0 (f 0)
    have h11 := hf y (-f y)
    have h12 := hf 0 (-f 0)
    have h13 := hf c (y - c)
    have h14 := hf 0 (y - c)
    have h15 := hf (y - c) c
    have h16 := hf (y - c) 0
    have h17 := hf (y - c) (f (y - c))
    have h18 := hf (y - c) (-f (y - c))
    norm_num [h_f0_eq_c, h_f_c_eq_c] at *
    <;>
    (try linarith) <;>
    (try nlinarith) <;>
    (try
      {
        nlinarith [sq_nonneg (f y - c), sq_nonneg (f 0), sq_nonneg (c), sq_nonneg
        ↪  (y), sq_nonneg (f c - c), sq_nonneg (f y)]
      }) <;>
    (try
      {
        nlinarith [sq_nonneg (f y - c), sq_nonneg (f 0), sq_nonneg (c), sq_nonneg
        ↪  (y), sq_nonneg (f c - c), sq_nonneg (f y), h_c_ge_zero]
      }) <;>
    (try
      {
        nlinarith [sq_nonneg (f y - c), sq_nonneg (f 0), sq_nonneg (c), sq_nonneg
        ↪  (y), sq_nonneg (f c - c), sq_nonneg (f y), h_c_ge_zero, sq_nonneg (f
y)]
      }) <;>
    (try
      {
        nlinarith [sq_nonneg (f y - c), sq_nonneg (f 0), sq_nonneg (c), sq_nonneg
        ↪  (y), sq_nonneg (f c - c), sq_nonneg (f y), h_c_ge_zero, sq_nonneg (f
y - c)]
      })

  have h_f_le_cy_add_c : ∀ y, f y ≤ c * y + c := by
    intro y
    have h₁ := h_f_le_c y
    have h₂ := h_f_le_c 0
```

```
    have h₃ := hf 0 y
    have h₄ := hf y 0
    have h₅ := hf y (-y)
    have h₆ := hf 0 (-y)
    have h₇ := hf y (c - y)
    have h₈ := hf 0 (c)
    have h₉ := hf c (-c)
    have h₁₀ := hf y 0
    have h₁₁ := hf 0 0
    have h₁₂ := hf y (f y)
    have h₁₃ := hf 0 (f 0)
    have h₁₄ := hf y (-f y)
    have h₁₅ := hf 0 (-f 0)
    have h₁₆ := hf c (y - c)
    have h₁₇ := hf 0 (y - c)
    have h₁₈ := hf (y - c) c
    have h₁₉ := hf (y - c) 0
    have h₂₀ := hf (y - c) (f (y - c))
    have h₂₁ := hf (y - c) (-f (y - c))
    norm_num [h_f0_eq_c, h_f_c_eq_c] at *
    <;>
    (try linarith) <;>
    (try nlinarith) <;>
    (try
      {
        nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y), sq_nonneg (f y + c - c * y)]
      }) <;>
    (try
      {
        nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y), sq_nonneg (f y + c - c * y), sq_nonneg (f y - c * y)]
      }) <;>
    (try
      {
        nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y), sq_nonneg (f y + c - c * y), sq_nonneg (f y - c * y), sq_nonneg
(f y - c)]
      })
    <;>
    (try
      {
        cases' le_total 0 y with hy hy <;>
        cases' le_total 0 (f y - c) with h h <;>
        cases' le_total 0 (c - y) with h' h' <;>
        nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y), sq_nonneg (f y + c - c * y), sq_nonneg (f y - c * y)]
      })
    <;>
    nlinarith

  exact ⟨h_c_ge_zero, h_f_c_eq_c, h_f_le_c, h_f_le_cy_add_c⟩

theorem imo2011_p3 (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x)) : ∀ x ≤ 0, f x = 0 := by
  -- Step 1: Prove that f is non-positive everywhere.
  have h_nonpos : ∀ x, f x ≤ 0 := aux_f_nonpositive f hf

  -- Step 2: Prove that there must exist some x₀ such that f(x₀) = 0.
  -- We prove this by contradiction. Assume f(x) is never zero.
  have h_exists_zero : ∃ x₀, f x₀ = 0 := by
    by_contra h_no_zero
    -- The hypothesis from `by_contra` is `h_no_zero : ¬(∃ x₀, f x₀ = 0)`.
```

```
    -- We use `push_neg` to convert it into a more usable form.
    push_neg at h_no_zero
    -- Now, `h_no_zero : ∀ (x : ℝ), f x ≠ 0`.

    -- This, combined with `h_nonpos`, implies f(x) < 0 for all x.
    have h_always_neg : ∀ x, f x < 0 := fun x ↦ (h_nonpos x).lt_of_ne (h_no_zero x)

    -- From this, we can deduce f(0) = f(f(0)).
    have h_f0_eq_ff0 : f 0 = f (f 0) := by
      have h_ff0_neg : f (f 0) < 0 := h_always_neg (f 0)
      have hle : f (f 0) ≤ f 0 := imo2011_p3_lemma1_f_neg_le_self f hf (f 0) h_ff0_neg
      have hge : f 0 ≤ f (f 0) := imo2011_p3_st1 f hf 0
      linarith

    -- Now, we use the main inequality to derive a contradiction.
    -- Let x = f(0) and y = -f(0).
    specialize hf (f 0) (-f 0)

    -- Define a local lemma for `a + -a = 0` to ensure it's available.
    have add_neg_self_local : f 0 + -f 0 = 0 := by ring

    -- Rewrite the inequality step-by-step to derive the contradiction.
    rw [add_neg_self_local] at hf
    rw [← h_f0_eq_ff0] at hf
    rw [← h_f0_eq_ff0] at hf

    -- The inequality is now f 0 ≤ -(f 0)² + f 0, which implies 0 ≤ -(f 0)².
    have h_contr : 0 ≤ -(f 0) ^ 2 := by linarith [hf]

    -- This is a contradiction because f(0) < 0, so -(f 0)² < 0.
    have h_f0_neg : f 0 < 0 := h_always_neg 0
    have h_sq_pos : 0 < (f 0) ^ 2 := sq_pos_of_ne_zero (ne_of_lt h_f0_neg)
    linarith

  -- Step 3: Use the existence of a zero to prove f(0) = 0.
  obtain ⟨x₀, hx₀⟩ := h_exists_zero
  have h_f0_eq_0 : f 0 = 0 := by
    -- A lemma gives properties of f if it has a zero. One is f(0) ≥ 0.
    have h_props := lemma_properties_if_f_has_zero f hf rfl ⟨x₀, hx₀⟩
    have h_f0_nonneg : f 0 ≥ 0 := h_props.1
    -- Combining f(0) ≥ 0 with f(0) ≤ 0 (from h_nonpos) gives f(0) = 0.
    linarith [h_nonpos 0]

  -- Step 4: Now that we have f(x) ≤ 0 and f(0) = 0, apply the final lemma.
  exact lemma_final_implication f hf h_f0_eq_0 h_nonpos
```

### E.4 IMO 2005 P3

```
-- Solution to IMO 2005 P3 by DRP-IMO

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat


theorem inequality_part1_nonnegative (x y z : ℝ) (hx : x > 0) (hy : y > 0) (hz : z > 0) (h : x * y * z ≥ 1) :
  (x*x - 1/x + y*y - 1/y + z*z - 1/z) / (x*x + y*y + z*z) ≥ 0 := by
```

```lean
  have h_main : x*x + y*y + z*z - (1/x + 1/y + 1/z) ≥ 0 := by
    have h₁ : 0 < x * y := by positivity
    have h₂ : 0 < x * z := by positivity
    have h₃ : 0 < y * z := by positivity
    have h₄ : 0 < x * y * z := by positivity
    have h₅ : 0 < x * y * z * x := by positivity
    have h₆ : 0 < x * y * z * y := by positivity
    have h₇ : 0 < x * y * z * z := by positivity
    field_simp [hx.ne', hy.ne', hz.ne']
    rw [le_div_iff₀ (by positivity)]
    -- Use nlinarith to prove the inequality
    nlinarith [sq_nonneg (x - y), sq_nonneg (x - z), sq_nonneg (y - z),
      sq_nonneg (x * y - 1), sq_nonneg (x * z - 1), sq_nonneg (y * z - 1),
      mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x - y)),
      mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x - z)),
      mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (y - z)),
      mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x * y - x * z)),
      mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x * y - y * z)),
      mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x * z - y * z))]

  have h_final : (x*x - 1/x + y*y - 1/y + z*z - 1/z) / (x*x + y*y + z*z) ≥ 0 := by
    have h₁ : x * x + y * y + z * z - (1 / x + 1 / y + 1 / z) ≥ 0 := h_main
    have h₂ : x * x + y * y + z * z > 0 := by positivity
    have h₃ : (x * x - 1 / x + y * y - 1 / y + z * z - 1 / z) / (x * x + y * y + z * z) ≥ 0 := by
      have h₄ : x * x - 1 / x + y * y - 1 / y + z * z - 1 / z = (x * x + y * y + z * z) - (1 / x + 1 / y + 1 / z) := by
        ring
      rw [h₄]
      have h₅ : ((x * x + y * y + z * z) - (1 / x + 1 / y + 1 / z)) / (x * x + y * y + z * z) ≥ 0 := by
        apply div_nonneg
        · linarith
        · linarith
      exact h₅
    exact h₃

  exact h_final


theorem inequality_part2_nonnegative (x y z : ℝ) (hx : x > 0) (hy : y > 0) (hz : z > 0) :
  ((x^5 - x^2)/(x^5 + y^2 + z^2) - (x*x - 1/x)/(x*x + y*y + z*z)) +
  ((y^5 - y^2)/(y^5 + z^2 + x^2) - (y*y - 1/y)/(y*y + z^2 + x*x)) +
  ((z^5 - z^2)/(z^5 + x^2 + y^2) - (z*z - 1/z)/(z*z + x*x + y*y)) ≥ 0 := by
  have h_main : ((x^5 - x^2)/(x^5 + y^2 + z^2) - (x*x - 1/x)/(x*x + y*y + z*z)) + ((y^5 - y^2)/(y^5 + z^2 + x^2) - (y*y - 1/y)/(y*y + z^2 + x*x)) + ((z
^5 - z^2)/(z^5 + x^2 + y^2) - (z*z - 1/z)/(z*z + x*x + y*y)) ≥ 0 := by
    have h₁ : (x^5 - x^2)/(x^5 + y^2 + z^2) - (x*x - 1/x)/(x*x + y*y + z*z) ≥ 0 := by
      have h₁₀ : 0 < x^5 + y^2 + z^2 := by positivity
      have h₁₁ : 0 < x*x + y*y + z*z := by positivity
      have h₁₂ : 0 < x^5 := by positivity
      have h₁₃ : 0 < x^3 := by positivity
      have h₁₄ : 0 < x^2 := by positivity
      have h₁₅ : 0 < x := by positivity
      field_simp
      rw [le_div_iff₀ (by positivity), ← sub_nonneg]
      ring_nf
      nlinarith [sq_nonneg (x^3 - x), sq_nonneg (x^2 - 1), sq_nonneg (x - 1),
        mul_nonneg hx.le (sq_nonneg (x^2 - 1)), mul_nonneg hx.le (sq_nonneg (x^3 - x)),
```

```
        mul_nonneg hx.le (sq_nonneg (x^2 - x)), mul_nonneg hx.le (sq_nonneg (x^3 - 1)),
        mul_nonneg (sq_nonneg (x - 1)) (sq_nonneg (x + 1)), mul_nonneg hx.le (sq_nonneg (x^2 - 2 * x + 1))]
    have h₂ : (y^5 - y^2)/(y^5 + z^2 + x^2) - (y*y - 1/y)/(y*y + z^2 + x*x) ≥ 0 := by
      have h₂₀ : 0 < y^5 + z^2 + x^2 := by positivity
      have h₂₁ : 0 < y*y + z^2 + x*x := by positivity
      have h₂₂ : 0 < y^5 := by positivity
      have h₂₃ : 0 < y^3 := by positivity
      have h₂₄ : 0 < y^2 := by positivity
      have h₂₅ : 0 < y := by positivity
      field_simp
      rw [le_div_iff₀ (by positivity), ← sub_nonneg]
      ring_nf
      nlinarith [sq_nonneg (y^3 - y), sq_nonneg (y^2 - 1), sq_nonneg (y - 1),
        mul_nonneg hy.le (sq_nonneg (y^2 - 1)), mul_nonneg hy.le (sq_nonneg (y^3 - y)),
        mul_nonneg hy.le (sq_nonneg (y^2 - y)), mul_nonneg hy.le (sq_nonneg (y^3 - 1)),
        mul_nonneg (sq_nonneg (y - 1)) (sq_nonneg (y + 1)), mul_nonneg hy.le (sq_nonneg (y^2 - 2 * y + 1))]
    have h₃ : (z^5 - z^2)/(z^5 + x^2 + y^2) - (z*z - 1/z)/(z*z + x*x + y*y) ≥ 0 := by
      have h₃₀ : 0 < z^5 + x^2 + y^2 := by positivity
      have h₃₁ : 0 < z*z + x*x + y*y := by positivity
      have h₃₂ : 0 < z^5 := by positivity
      have h₃₃ : 0 < z^3 := by positivity
      have h₃₄ : 0 < z^2 := by positivity
      have h₃₅ : 0 < z := by positivity
      field_simp
      rw [le_div_iff₀ (by positivity), ← sub_nonneg]
      ring_nf
      nlinarith [sq_nonneg (z^3 - z), sq_nonneg (z^2 - 1), sq_nonneg (z - 1),
        mul_nonneg hz.le (sq_nonneg (z^2 - 1)), mul_nonneg hz.le (sq_nonneg (z^3 - z)),
        mul_nonneg hz.le (sq_nonneg (z^2 - z)), mul_nonneg hz.le (sq_nonneg (z^3 - 1)),
        mul_nonneg (sq_nonneg (z - 1)) (sq_nonneg (z + 1)), mul_nonneg hz.le (sq_nonneg (z^2 - 2 * z + 1))]
    linarith
  exact h_main


-- The main theorem
theorem imo2005_p3 (x y z : ℝ) (hx : x > 0) (hy : y > 0) (hz : z > 0) (h_prod_ge_1 : x * y * z ≥ 1) :
  (x ^ 5 - x ^ 2) / (x ^ 5 + y ^ 2 + z ^ 2) + (y ^ 5 - y ^ 2) / (y ^ 5 + z ^ 2 + x ^ 2) + (z ^ 5 - z ^ 2) / (z ^ 5 + x ^ 2 + y ^ 2) ≥ 0 := by
  -- Define S_part1 (LHS of inequality_part1_nonnegative)
  let S_part1 := (x^2 - 1/x + y^2 - 1/y + z^2 - 1/z) / (x^2 + y^2 + z^2)

  -- Define S_part2 (LHS of inequality_part2_nonnegative)
  let S_part2 :=
    ((x^5 - x^2)/(x^5 + y^2 + z^2) - (x^2 - 1/x)/(x^2 + y^2 + z^2)) +
    ((y^5 - y^2)/(y^5 + z^2 + x^2) - (y^2 - 1/y)/(y^2 + z^2 + x^2)) +
    ((z^5 - z^2)/(z^5 + x^2 + y^2) - (z^2 - 1/z)/(z^2 + x^2 + y^2))

  -- Prove S_part1 ≥ 0 using inequality_part1_nonnegative
  have h_S_part1_nonneg : S_part1 ≥ 0 := by
    apply inequality_part1_nonnegative <;> assumption

  -- Prove S_part2 ≥ 0 using inequality_part2_nonnegative
```

```lean
  have h_S_part2_nonneg : S_part2 ≥ 0 := by
    apply inequality_part2_nonnegative <;> assumption

  -- Prove that the original LHS is equal to S_part1 + S_part2
  -- We'll prove it as a separate fact (have) and then use it.
  have h_LHS_eq_sum :
    (x^5 - x^2)/(x^5 + y^2 + z^2) + (y^5 - y^2)/(y^5 + z^2 + x^2) + (z^5 - z^2)/(z^5 + x^2 + y^2) =
    S_part2 + S_part1 := by
    -- Expand the definitions of S_part1 and S_part2
    unfold S_part1 S_part2
    -- Normalize denominators which are permutations of each other
    have h_denom_y : y^2 + z^2 + x^2 = x^2 + y^2 + z^2 := by ac_rfl
    have h_denom_z : z^2 + x^2 + y^2 = x^2 + y^2 + z^2 := by ac_rfl
    rw [h_denom_y, h_denom_z]
    -- The rest is a pure algebraic identity, which `ring` can solve.
    -- It correctly rearranges terms like (a-b)+(c-d)+(e-f) + (b+d+f)/k = a+c+e
    -- after combining the fractions for S_part1
    ring

  -- Rewrite the goal using the equality we just proved
  rw [h_LHS_eq_sum]

  -- The goal is now S_part2 + S_part1 ≥ 0, which follows from the two parts being non-negative.
  exact add_nonneg h_S_part2_nonneg h_S_part1_nonneg
```

### E.5 IMO 2000 P2

```lean
-- Solution to IMO 2000 P2 by DRP-IMO

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/--Given positive real numbers \( a \), \( b \), and \( c \) such that \( a \times b \times c = 1 \), prove that there exist positive real numbers \( x
 \), \( y \), and \( z \) such that \( a = \frac{x}{y} \), \( b = \frac{y}{z} \), and \( c = \frac{z}{x} \).-/
theorem imo2000_p2_existence_of_xyz (a b c : ℝ) (ha : 0 < a) (hb : 0 < b) (hc : 0 < c) (habc : a * b * c = 1) :
  ∃ x y z : ℝ, 0 < x ∧ 0 < y ∧ 0 < z ∧ a = x/y ∧ b = y/z ∧ c = z/x := by
  have h_main : ∃ (x y z : ℝ), 0 < x ∧ 0 < y ∧ 0 < z ∧ a = x/y ∧ b = y/z ∧ c = z/x := by
    refine' ⟨a, 1, 1 / b, _, _, _, _, _, _⟩
    · -- Prove that a > 0
      linarith
    · -- Prove that 1 > 0
      norm_num
    · -- Prove that 1 / b > 0
      exact div_pos zero_lt_one hb
    · -- Prove that a = a / 1
      field_simp
    · -- Prove that b = 1 / (1 / b)
      field_simp
      <;>
      nlinarith
    · -- Prove that c = (1 / b) / a
      have h₁ : c = 1 / (a * b) := by
        have h₂ : a * b * c = 1 := habc
```

```lean
        have h₃ : c = 1 / (a * b) := by
          have h₄ : a * b ≠ 0 := by positivity
          field_simp [h₄] at h₂ ⊢
          nlinarith
        exact h₃
      have h₂ : (1 / b : ℝ) / a = 1 / (a * b) := by
        field_simp
        <;> ring
        <;> field_simp [ha.ne', hb.ne']
        <;> nlinarith
      rw [h₁] at *
      <;> linarith
  exact h_main

/--Consider three positive real numbers \( x \), \( y \), and \( z \) such that \( x > 0 \), \( y > 0 \), and \( z > 0 \). Prove that the product of th
e expressions \( (x - y + z) \), \( (y - z + x) \), and \( (z - x + y) \) is less than or equal to the product \( x \cdot y \cdot z \).-/
theorem schur_like_ineq (x y z : ℝ) (hx : 0 < x) (hy : 0 < y) (hz : 0 < z) :
  (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
  have h_main : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
    nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
      mul_nonneg hx.le hy.le, mul_nonneg hy.le hz.le, mul_nonneg hz.le hx.le,
      mul_nonneg (sq_nonneg (x - y)) (sq_nonneg (y - z)),
      mul_nonneg (sq_nonneg (y - z)) (sq_nonneg (z - x)),
      mul_nonneg (sq_nonneg (z - x)) (sq_nonneg (x - y)),
      mul_nonneg (sq_nonneg (x - y + z)) (sq_nonneg (y - z + x)),
      mul_nonneg (sq_nonneg (y - z + x)) (sq_nonneg (z - x + y)),
      mul_nonneg (sq_nonneg (z - x + y)) (sq_nonneg (x - y + z)),
      mul_nonneg (sq_nonneg (x + y - z)) (sq_nonneg (y + z - x)),
      mul_nonneg (sq_nonneg (y + z - x)) (sq_nonneg (z + x - y)),
      mul_nonneg (sq_nonneg (z + x - y)) (sq_nonneg (x + y - z))]
  exact h_main

/--Consider positive real numbers \( a, b, c, x, y, z \) such that \( a \cdot b \cdot c = 1 \) and \( a = \frac{x}{y} \), \( b = \frac{y}{z} \), \( c =
 \frac{z}{x} \). Prove that the inequality \((a - 1 + \frac{1}{b}) \cdot (b - 1 + \frac{1}{c}) \cdot (c - 1 + \frac{1}{a}) \leq 1\) is equivalent to th
e inequality \((x - y + z) \cdot (y - z + x) \cdot (z - x + y) \leq x \cdot y \cdot z\).-/
theorem inequality_equivalence_under_parametrization (a b c x y z : ℝ)
  (ha : 0 < a) (hb : 0 < b) (hc : 0 < c) (habc : a * b * c = 1)
  (hx : 0 < x) (hy : 0 < y) (hz : 0 < z)
  (hax : a = x / y) (hby : b = y / z) (hcz : c = z / x) :
  (a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) ≤ 1 ↔
  (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
  have h_main : (a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) = ((x + z - y) / y) * ((x + y - z) / z) * ((y + z - x) / x) := by
    have h₁ : a - 1 + 1 / b = (x + z - y) / y := by
      have h₁ : a = x / y := by linarith
      have h₂ : b = y / z := by linarith
      rw [h₁, h₂]
      field_simp [ha.ne', hb.ne', hx.ne', hy.ne', hz.ne']
      <;> ring_nf
      <;> field_simp [ha.ne', hb.ne', hx.ne', hy.ne', hz.ne']
      <;> nlinarith
    have h₂ : b - 1 + 1 / c = (x + y - z) / z := by
      have h₁ : b = y / z := by linarith
      have h₂ : c = z / x := by linarith
      rw [h₁, h₂]
      field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
      <;> ring_nf
      <;> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
```

```lean
      <;> nlinarith
    have h₃ : c - 1 + 1 / a = (y + z - x) / x := by
      have h₁ : c = z / x := by linarith
      have h₂ : a = x / y := by linarith
      rw [h₁, h₂]
      field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
      <;> ring_nf
      <;> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
      <;> nlinarith
    rw [h₁, h₂, h₃]
    <;> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
    <;> ring_nf
    <;> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
    <;> nlinarith

  have h_equiv : ((x + z - y) / y) * ((x + y - z) / z) * ((y + z - x) / x) ≤ 1 ↔ (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
    have h₁ : 0 < x * y := by positivity
    have h₂ : 0 < y * z := by positivity
    have h₃ : 0 < z * x := by positivity
    have h₄ : 0 < x * y * z := by positivity
    constructor
    · intro h
      have h₅ : ((x + z - y) / y) * ((x + y - z) / z) * ((y + z - x) / x) ≤ 1 := by linarith
      have h₆ : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
        field_simp at h₅
        rw [div_le_one (by positivity)] at h₅
        nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
          mul_nonneg hx.le hy.le, mul_nonneg hy.le hz.le, mul_nonneg hz.le hx.le,
          mul_nonneg (sq_nonneg (x - y)) hz.le, mul_nonneg (sq_nonneg (y - z)) hx.le,
          mul_nonneg (sq_nonneg (z - x)) hy.le]
      linarith
    · intro h
      have h₅ : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by linarith
      have h₆ : ((x + z - y) / y) * ((x + y - z) / z) * ((y + z - x) / x) ≤ 1 := by
        field_simp
        rw [div_le_one (by positivity)]
        nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
          mul_nonneg hx.le hy.le, mul_nonneg hy.le hz.le, mul_nonneg hz.le hx.le,
          mul_nonneg (sq_nonneg (x - y)) hz.le, mul_nonneg (sq_nonneg (y - z)) hx.le,
          mul_nonneg (sq_nonneg (z - x)) hy.le]
      linarith

  have h_final : (a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) ≤ 1 ↔ (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
    rw [h_main]
    rw [h_equiv]
    <;>
    simp_all
    <;>
    field_simp
    <;>
    ring_nf
    <;>
    nlinarith

  exact h_final

theorem imo2000_p2
    (a b c : ℝ) (ha : 0 < a) (hb : 0 < b) (hc : 0 < c)
```

```
  (habc : a * b * c = 1) :
  (a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) ≤ 1 := by
-- 1. Parametrize x y z using positive numbers
obtain ⟨x, y, z, hx, hy, hz, ha_eq, hb_eq, hc_eq⟩ :=
  imo2000_p2_existence_of_xyz a b c ha hb hc habc
-- 2. Use an equivalent lemma to transform the goal into the form involving x y z
have h_equiv :=
  (inequality_equivalence_under_parametrization
      (a := a) (b := b) (c := c) (x := x) (y := y) (z := z)
      ha hb hc habc hx hy hz ha_eq hb_eq hc_eq)
-- 3. The Schur-type inequality yields the conclusion on the right-hand side.
have hxyz : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z :=
  schur_like_ineq x y z hx hy hz
-- 4. Derive the original conclusion by reversing the equivalent proposition.
exact h_equiv.mpr hxyz
```

## F Prompts Used for Step3

**Prompt for Final Proof Generation (Step3)**

This is a very challenging Lean 4 proof problem that is difficult to solve directly.
To assist with the proof, I have provided several already-proven sub-theorems.
Please plan how to use these sub-theorems to construct a complete proof of the main theorem.
Use the provided sub-theorems freely (their proofs are marked with ‘sorry‘), but **you must not use ‘sorry‘ anywhere else** in your final proof.
Finally, output the complete Lean 4 code for the proof.

**Main theorem:**

{problem}

**Proved Sub-theorems:**

{sub-theorems}