

# Towards Solving More Challenging IMO Problems via Decoupled Reasoning and Proving

Zhenwen Liang, Linfeng Song, Yang Li, Tao Yang, Feng Zhang, Haitao Mi and Dong Yu

Tencent AI Lab

{zhenwzliang, lfsong, haitaomi}@global.tencent.com

## 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 release our full dataset of generated and verified lemmas for a wide range of IMO problems, available at <https://tencent-imo.github.io/>.

## 1 Introduction

Automated Theorem Proving (ATP) is the task of automatically generating formal proofs for mathematical or logical statements. By translating problems into a formal language (e.g., Lean (Moura & Ullrich, 2021) or Isabelle (Paulson, 1994)) and iteratively applying tactics within a proof assistant’s environment, an ATP system can construct machine-verified proofs that guarantee logical correctness. This verifiability makes ATP indispensable for the formal verification of critical software and hardware systems, where every reasoning step must be rigorously checked. ATP has long been a foundational challenge in both AI and mathematics, as such systems could leverage massive computational power to help mathematicians evaluate and even solve *open conjectures*. Recent breakthroughs in large language models (LLMs) have catalyzed rapid progress in ATP. Leveraging techniques such as expert iteration (Polu & Sutskever, 2020), tree search (Li et al., 2024; Wu et al., 2024; Xin et al., 2025; Liang et al., 2025), chain-of-thought reasoning (Xin et al., 2024; Lin et al., 2025), and reinforcement learning, state-of-the-art provers have achieved remarkable performance gains.

However, recent research in the LLM reasoning community has exposed a significant and widening gap between informal reasoning and formal proving capabilities. For instance, a recent evaluation (Dekoninck et al., 2025) on the PutnamBench illustrates this chasm: *Top-tier LLMs like Gemini 2.5 Pro can generate informal, natural-language solutions with over 80% accuracy after human verification. In contrast, the best formal prover Deepseek-Prover-V2 671B struggles to solve even 8% of the same problems directly.* This highlights a critical dilemma for the field: while LLMs possess immense mathematical reasoning power, their informal outputs lack the rigor and machine-verifiable guarantees required for formal correctness. Conversely, formal ATP systems have this guarantee but lack the raw

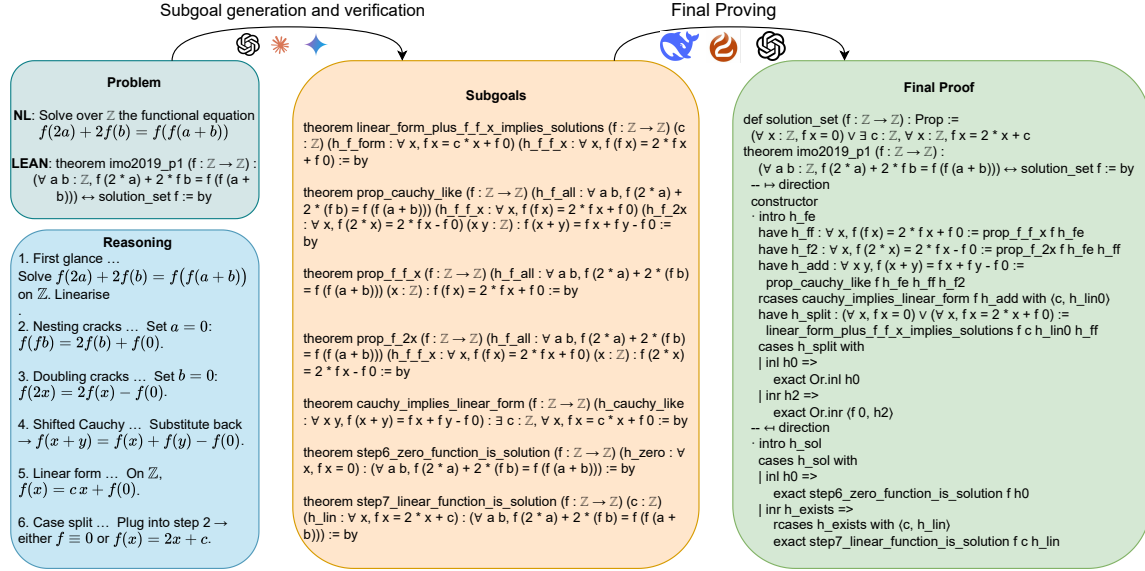


Figure 1: The overall pipeline of DRP-IMO taking the problem of IMO 2019 P1 as an example. Detailed proofs of the subgoals are omitted for brevity.

problem-solving ability to tackle complex challenges. Bridging this gap is arguably the most pressing challenge in modern ATP. To address this, state-of-the-art provers like DeepSeek-Prover-v2 (Ren et al., 2025) and Kimina (Wang et al., 2025) attempt to integrate reasoning and proving. They generate high-level sketches or subgoals before producing the final formal proof. However, we argue that these approaches suffer from a fundamental design flaw: by tightly coupling high-level planning and low-level proof generation within a single, monolithic model, they inadvertently shackle the model’s full reasoning potential. Its “sketching” is constrained by what its integrated proving component can handle. This inherent limitation prevents them from fully leveraging the powerful informal reasoning capabilities demonstrated on benchmarks like PutnamBench. This, in turn, explains why even these advanced systems still fail on the most difficult problems, such as post-2000 International Mathematical Olympiad (IMO) problems.

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), the 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, omega, or try. This over-reliance is not merely a shortcut, but a symptom of a training-induced degradation of its reasoning capabilities, effectively preventing it from generating the insightful decompositions required for truly difficult problems.

Based on this diagnosis, we propose a new framework built on a foundational principle: the decoupling of high-level reasoning from low-level proof generation. We argue that to bridge the chasm between informal insight and formal correctness, the two processes must be handled by distinct, specialized models scheduled with greater flexibility. Our approach leverages a powerful, general-purpose LLM as a dedicated *Reasoner* and a separate, efficient ATP model as the *Prover*. This architectural separation liberates the *Reasoner* to focus exclusively on its strength—generating high-level mathematical insights and strategic decompositions—while the *Prover* ensures that the

resulting strategy can be grounded in formal, verifiable logic. It is this principled, decoupled bridge that we believe can finally harness the reasoning power of LLMs for formal theorem proving.

Our simple yet effective pipeline is illustrated in Figure 1. Given a theorem, the Reasoner is first invoked to propose potentially useful lemmas (subgoals), expressed only as formal statements, which act as a bridge between high-level strategy and formal proof. A Prover module then attempts to verify these proposed lemmas, filtering out any that are unprovable. Finally, the Prover tackles the main theorem, now armed with a set of verified lemmas that guide the proof search and significantly reduce its complexity. This contrasts with existing approaches that follow a rigid, one-shot "reasoning-then-proving" trajectory within a single model.

We evaluate our approach on a challenging set of non-geometry IMO problems from 2000 to 2024. To the best of our knowledge, no existing open-source automated theorem prover has reported success on any problem from this set. In the experiments, our method successfully solves 5 of these problems: IMO 2000 Problem 2, IMO 2005 Problem 3, IMO 2011 Problem 3, IMO 2019 Problem 1, and IMO 2020 Problem 2.

## 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 is 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, can be found at <https://tencent-imo.github.io/>.

## 2 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

```

1  import Mathlib
2  import Aesop
3  set_option maxHeartbeats 0
4  open BigOperators Real Nat Topology Rat
5
6  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
7      have hb : b = a + 1 := h₁
8      have hc : c = a + 2 := by omega
9      rw [hb, hc] at h₃
10     have h4 : a = 4 := by
11         have h_pos : 0 < a := h₀.left
12         have : a ≤ 6 := by
13             nlinarith [h₃, mul_pos h_pos (show 0 < a + 1 by omega), mul_pos h_pos (show 0
               ↪ < a + 2 by omega),
14                 show 0 < a + 1 by omega, show 0 < a + 2 by omega]
15         interval_cases a <|> omega
16     have ha : a = 4 := h4
17     have hb' : b = 5 := by omega
18     have hc' : c = 6 := by omega
19     rw [ha, hb', hc']
20     norm_num

```

Figure 2: Kimina solutions to amc12b\_2002\_p7. The have statements hb and hc are trivially extended from provided conditions, and statement h4 makes a luckily guess that directly leads to the solution. Automatic tactic omega plays a central role throughout the proof.

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.

### 3 A Framework for Decoupled Reasoning and Proving

Our methodology is founded on the principle of decoupling high-level strategic reasoning from low-level formal proof generation. This separation allows us to use the best tool for each task: a powerful, general-purpose reasoning model for strategic decomposition, and a specialized, efficient theorem prover for formal verification. The overall workflow, illustrated in Figure 1, consists of three main stages: subgoal generation, subgoal filtering, and final proof construction.

#### 3.1 Stage 1: Strategic Subgoal Generation with a Reasoner

The first stage aims to replicate the most crucial aspect of human mathematical problem-solving: identifying strategic intermediate steps or lemmas. For IMO-level problems, which can require hundreds of proof steps in a formal language like Lean, a brute-force search is intractable. A well-chosen set of lemmas can dramatically prune this search space by decomposing the primary goal into more manageable components.

Our central insight is to leverage a powerful, general-purpose Large Language Model as a dedicated *Reasoner*, whose sole responsibility is to generate these strategic decompositions. Unlike specialized ATP models trained primarily on code, our *Reasoners* (e.g., OpenAI-o3, Gemini 2.5, Claude 4 Opus) excel at high-level, semantic understanding and creative problem-solving. We task the Reasoner with generating only the \*formal statements of potential lemmas, without their proofs. This deliberate constraint focuses the model on its core strength—strategic thinking—while avoiding the complexities and potential errors of full proof generation.

After evaluating several state-of-the-art models, including OpenAI-o3, Claude 4 Opus and Gemini 2.5 Pro, we selected Gemini 2.5 Pro for its superior ability to generate diverse and logically sound subgoals for complex mathematical problems. We use the following prompt, which is designed to encourage deep reasoning before outputting structured Lean statements:

##### Prompt for Subgoal Generation

You are given a very challenging theorem written in Lean 4. This theorem is too difficult to prove directly. Your task is to think step-by-step to devise a feasible and complete proof strategy for the theorem, and then decompose the original theorem into a sequence of smaller, logically coherent sub-theorems, each of which can be proved more easily.

Important instructions:

First, reason through and construct a valid and complete proof strategy for the original theorem.

After the solution path is clear, divide it into intermediate proof steps. Each step should be expressed as a separate sub-theorem in Lean 4, following the same syntactic and semantic format as the original theorem.

The decomposition should reflect deep understanding of the overall proof structure. Avoid trivial splits such as case analysis or mechanical “divide into two cases” tactics unless they are genuinely part of the reasoning process.

Each sub-theorem must represent a meaningful proof milestone — essentially a condensed logical step from the overall proof strategy.

The sub-theorems should be self-contained and provable, and collectively they should imply the original theorem.

Output format:

A brief explanation of your proof strategy (in natural language or Lean comments).

A list of Lean 4 theorem declarations, each representing a sub-theorem, all starting with 'theorem XXX' and ending with ':= by sorry'.

Ensure all sub-theorems are expressed using the same formal syntax and conventions as the input theorem.

Input Theorem:

To reliably extract the generated lemma statements from the model's free-form text output, we apply a simple yet robust regular expression. This ensures a clean handoff to the next stage of the pipeline.

#### Regular Expression for Lemma Extraction

```
theorem (.*) := by sorry
```

### 3.2 Stage 2: Subgoal Verification and Filtering

The second stage acts as a critical filter to ensure that only logically sound and verifiable lemmas are passed to the final stage. This verification step is essential for the overall soundness of our framework. For this task, we employ a dedicated *Prover* model. The key requirement for this *Prover* is not high-level reasoning, but rather strong tactical execution and efficiency in proving well-defined, modular goals. We selected DeepSeek-Prover-v2 (7B, CoT version) for this role, as it offers a state-of-the-art balance of proof success rate and computational efficiency. Each candidate lemma generated in Stage 1 is treated as an independent theorem. The *Prover* attempts to find a proof for it, generating up to  $k$  candidate proofs. A lemma is considered "verified" and retained if at least one attempt succeeds.

This filtering process is not merely a correctness check; it is a core component of our decoupled strategy. It allows the *Reasoner* in Stage 1 to be "creative" and even "speculative," proposing diverse ideas without the immediate burden of provability. The *Prover* then grounds this creativity in formal rigor, effectively selecting the most promising pathways. We set  $k$  (the number of proof candidates) to 128, a value chosen empirically to balance the exploration breadth against computational cost.

### 3.3 Stage 3: Final Proof Construction

In the final stage, the prover attempts to solve the main theorem by leveraging the set of verified lemmas obtained in Stage 2. These lemmas are prepended to the context of the main problem statement, effectively enriching the problem statement with a suite of pre-proven, reusable components. This process transforms the original, monolithic proof task into a more tractable task of assembling a final proof using these intermediate results as foundational building blocks, as illustrated in Figure 1.

In the final stage, the prover tries to tackle the main theorem with the aid of the set of verified lemmas from Stage 2. These lemmas are prepended to the context of the main problem statement, effectively enriching the problem with powerful, pre-proven tools. This transforms the original, monolithic challenge into a simpler task of assembling a final proof using these new building blocks, as illustrated in Figure 1.

A crucial challenge we identified at this stage 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 that not all provers possess equally. After further experimentation, we found that two powerful reasoner models, OpenAI-o3 and Gemini 2.5 Pro, are significantly more adept at integrating and applying the provided lemmas to construct the final proof. This highlights the importance of selecting the right tool for each sub-task in a decoupled system. While we use those reasoners for our final results, a promising direction for future work is to fine-tune specialized provers like DeepSeek-Prover-v2 on a curriculum of problems that explicitly require the use of given lemmas.

## 4 Experiment

We evaluate our approach on challenging problems from the International Mathematical Olympiad (IMO), focusing on non-geometry problems from the years 2000 to 2024. Each annual IMO consists of six problems, typically including one geometry problem. We focus on the non-geometry ones, and our method successfully solves 5 of these problems: IMO 2000 Problem 2, IMO 2005 Problem 3, IMO 2011 Problem 3, IMO 2019 Problem 1, and IMO 2020 Problem 2. We list detailed proofs in Appendix B and show current progress on IMO 2024 problems in Appendix A.

### 4.1 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  $f : \mathbb{Z} \rightarrow \mathbb{Z}$  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.

#### 4.1.1 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.

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 h29 : f x = 2 * x + f 0 := by
  have h291 := hf x 0
  ...
  ring_nf at h291 h292 ... ⊢
<;> 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 h31 : f x = 2 * x + (f 0 - 2 * 0) := by
  have h32 := hf 0 0
  ...
  have h42 := hf 1 (x - 1)
  ring_nf at h32 h33 ... h42 ⊢
  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.

## 4.2 Degradation of Mathematical Reasoning in Specialised Provers

**Motivation.** A central hypothesis of our work is that the prevailing reinforcement learning with verifiable rewards (RLVR) paradigm, while effective for optimizing success rates on specific ATP benchmarks, may inadvertently cause a degradation in the model’s intrinsic mathematical reasoning capabilities. The reward signal, being solely dependent on the formal proof’s success, does not explicitly value the quality or correctness of the natural language reasoning that may precede it. To test this hypothesis, we designed an experiment to isolate and measure this potential degradation.

**Experimental Setup.** We compare the performance of a specialized prover model with its general-purpose base model on standard mathematical reasoning benchmarks that do not involve formal proof generation. Specifically, we selected:

- **Base Model:** Qwen2.5-Math-7B-Instruct, the foundational model upon which the Kimina-Prover is built, which is highly capable in general mathematical problem-solving.
- **Prover:** Kimina-Prover-Preview-Distill-7B, a state-of-the-art prover initialized from Qwen2.5-Math-7B-Instruct and fine-tuned for Lean-based theorem proving.

We evaluated both models on the **MATH** and **AIME** benchmarks. We found that with appropriate prompting, Kimina-Prover can still generate high-quality solutions to math problems, rather than generating Lean code. This allowed for a direct comparison of their problem-solving accuracy. We report the pass@k accuracy for both models on AIME24.

Table 1: Performance comparison on general mathematical reasoning benchmarks.

Model	MATH	AIME24			
	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%
<b>Performance Drop (pts)</b>	<b>-4.9</b>	<b>-5.7</b>	<b>-9.2</b>	<b>-11.3</b>	<b>-5.8</b>

The results, presented in Table 1, provide clear evidence supporting our hypothesis. On both benchmarks, the Kimina-Prover exhibits a marked decline in performance compared to its base model. The single-attempt accuracy (pass@1) drops by 4.9 percentage points on MATH and 5.7 points on AIME. Crucially, this performance gap is not an isolated phenomenon but persists robustly across multi-sample evaluations on the challenging AIME dataset. The performance delta remains substantial at pass@4, widens further at pass@8, and is still significant at pass@16. This confirms that the specialization process for formal theorem proving, while boosting performance on ATP tasks, comes at the cost of broader mathematical reasoning skills. This finding strongly motivates our decoupled approach: instead of attempting to force a single model to excel at both high-level reasoning and low-level formalization, we should leverage a dedicated, un-degraded reasoning model for the former, preserving its full intellectual capacity.

## 4.3 Further Discussions

**On the Utilization of External Knowledge: Lemmas vs. have Statements.** A critical challenge we encountered during the development of our pipeline was the effective integration of verified subgoals into the final proof stage. We observed a significant behavioral pattern: when verified

subgoals were provided as standalone lemmas in the context, many state-of-the-art provers, including DeepSeek-Prover-v2, tended to ignore them. Instead of leveraging these pre-proven facts, the models often attempted to prove the main theorem from scratch, indicating a form of "contextual blindness" or a bias towards self-contained proof generation learned during their fine-tuning.

The "contextual blindness" phenomenon contrasts with using a `have` statement within a proof, which forces the model to work with a specific, locally-defined fact (Ren et al., 2025; Cao et al., 2025). However, `have` statements require strict alignment of symbols and definitions with the current proof state, limiting their flexibility. Standalone lemmas, in theory, offer a more powerful and flexible mechanism for incorporating external knowledge, as they do not impose such rigid constraints. This flexibility is crucial for solving complex problems where reusing established results is key. Our findings suggest that a significant gap exists in the ability of current provers to effectively utilize modular, pre-proven knowledge. A crucial direction for future work is therefore to develop or fine-tune provers to specifically excel at this "proof continuation" task, enabling them to robustly accept and leverage a library of existing theorems and lemmas.

**Limitations and Failure Analysis.** Our analysis on unsolved statements reveals two primary bottlenecks in the current pipeline, highlighting the remaining gap between our automated system and human-level mathematical ingenuity.

First, the primary mode of failure is the Prover’s inability to verify critical, complex lemmas. For many unsolved problems, our Reasoner successfully identified a plausible high-level strategy, but the constituent lemmas were simply too difficult for the Prover module to handle. To validate this, we conducted an oracle experiment where we manually proved these bottleneck lemmas (or replaced their proofs with `sorry`). In this idealized setting, our framework was able to solve a significantly larger number of IMO problems. This demonstrates that the performance of the entire pipeline is currently bounded by the raw theorem-proving power of the Prover component. Another potential approach involves further decomposing unresolved lemmas into simpler sub-lemmas to facilitate their proof.

Second, we identified a fundamental difference in the reasoning style of our Reasoner compared to human mathematicians. Through manual inspection of our Reasoner’s outputs against official IMO solutions, we observed that human proofs often hinge on a single, "magical" insight or a clever re-framing of the problem that dramatically simplifies the proof. Our Reasoner, reflecting its LLM nature, excels at systematic, step-by-step decomposition and logical deduction. It is proficient at breaking a problem down into a clear chain of sub-problems but struggles to generate the kind of non-obvious, highly creative leaps that characterize elegant human solutions. This "ingenuity gap" represents a deeper, more fundamental challenge for LLM-based reasoning and is a key bottleneck for our framework on the most difficult problems.

## 5 Conclusion

In this work, we introduce a novel framework for automated theorem proving that addresses a core limitation in current systems: the degradation of mathematical reasoning ability caused by end-to-end RLVR training. Our central contribution is the principle of decoupling strategic reasoning from formal proof generation. We achieve this by delegating high-level strategic thinking to a dedicated Reasoner—a powerful, general-purpose LLM whose nuanced reasoning capabilities are often compromised during specialized prover fine-tuning. This Reasoner formulates its strategy as a set of formal subgoal lemmas, providing a more general and powerful mechanism for problem decomposition than restrictive in-proof statements. This modular design ensures proof search is guided by a coherent, human-like plan.

Our evaluation on a challenging set of post-2000 IMO problems, where we successfully solved 5 problems previously unsolved by any open-source prover, provides strong evidence for the efficacy of our decoupled approach. We acknowledge that our current evaluation is limited and that the multi-stage nature of our pipeline presents scalability challenges. A key priority for future work

is to streamline this pipeline to enable comprehensive evaluation on large-scale benchmarks such as miniF2F and ProofNet. Ultimately, we believe that by separating the art of strategy from the science of verification, our work paves the way for more robust, scalable, and insightful automated reasoning systems capable of tackling the frontiers of mathematics.

## References

- Chenrui Cao, Liangcheng Song, Zenan Li, Xinyi Le, Xian Zhang, Hui Xue, and Fan Yang. Reviving dsp for advanced theorem proving in the era of reasoning models. *arXiv preprint arXiv:2506.11487*, 2025.
- 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*, pp. 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>.
- Yang Li, Dong Du, Linfeng Song, Chen Li, Weikang Wang, Tao Yang, and Haitao Mi. Hunyuanprover: A scalable data synthesis framework and guided tree search for automated theorem proving. *arXiv preprint arXiv:2412.20735*, 2024.
- Zhenwen Liang, Linfeng Song, Yang Li, Tao Yang, Feng Zhang, Haitao Mi, and Dong Yu. Mps-prover: Advancing stepwise theorem proving by multi-perspective search and data curation. *arXiv preprint arXiv:2505.10962*, 2025.
- Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia, Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated theorem proving. *arXiv preprint arXiv:2502.07640*, 2025.
- Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language. In *Automated Deduction—CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pp. 625–635. Springer, 2021.
- Lawrence C Paulson. *Isabelle: A generic theorem prover*. Springer, 1994.
- Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.
- ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanbiao 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=yAa5192TtQ>.
- 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.

Zijian Wu, Suozhi Huang, Zhejian Zhou, Huaiyuan Ying, Jiayu Wang, Dahua Lin, and Kai Chen. Internlm2.5-stepprover: Advancing automated theorem proving via expert iteration on large-scale lean problems. *arXiv preprint arXiv:2410.15700*, 2024.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanxia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024.

Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving. *arXiv preprint arXiv:2502.03438*, 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 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.

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

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

#### A.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
```

### A.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.

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

### A.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.

### A.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
```



### A.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.

## B Solved IMO problems

### B.1 IMO 2020 P2

```
1  -- Solution to IMO 2020 P2 by DRP-IMO
2
3  import Mathlib
4  import Aesop
5
6  set_option maxHeartbeats 0
7
8  open BigOperators Real Nat Topology Rat
9
10 /--Consider four real numbers  $\mid (a, b, c, \mid)$  and  $\mid (d \mid)$  such that  $\mid (0 < d \mid \leq c \mid \leq b$ 
     $\rightarrow \mid \leq a \mid)$  and their sum is equal to 1, i.e.,  $\mid (a + b + c + d = 1 \mid)$ . Prove that the
     $\rightarrow$  product of the weighted sum  $\mid (a + 2b + 3c + 4d \mid)$  and the sum of their squares  $\mid (a^2$ 
     $\rightarrow + b^2 + c^2 + d^2 \mid)$  is less than 1.-/
```

```

11 theorem weighted_sum_times_sum_sq_lt_one (a b c d : ℝ) (hd_pos : 0 < d) (hdc : d ≤ c)
12   → (hcb : c ≤ b) (hba : b ≤ a) (h_sum_eq_1 : a + b + c + d = 1) :
13     (a + 2*b + 3*c + 4*d) * (a^2 + b^2 + c^2 + d^2) < 1 := by
14       have h_a_lt_one : a < 1 := by
15         have h1 : a + b + c + d = 1 := h_sum_eq_1
16         have h2 : 0 < b := by linarith
17         have h3 : 0 < c := by linarith
18         have h4 : 0 < d := hd_pos
19         nlinarith
20       have h_b_lt_one : b < 1 := by
21         have h1 : a ≥ b := by linarith
22         have h2 : a < 1 := h_a_lt_one
23         nlinarith
24       have h_c_lt_one : c < 1 := by
25         have h1 : b ≥ c := by linarith
26         have h2 : b < 1 := h_b_lt_one
27         nlinarith
28       have h_d_lt_one : d < 1 := by
29         have h1 : c ≥ d := by linarith
30         have h2 : c < 1 := h_c_lt_one
31         nlinarith
32       have h_a2_lt_a : a^2 < a := by
33         have h1 : a < 1 := h_a_lt_one
34         have h2 : a > 0 := by linarith
35         nlinarith [sq_pos_of_pos h2]
36         <=> nlinarith
37       have h_b2_lt_b : b^2 < b := by
38         have h1 : b < 1 := h_b_lt_one
39         have h2 : b > 0 := by linarith
40         nlinarith [sq_pos_of_pos h2]
41         <=> nlinarith
42       have h_c2_lt_c : c^2 < c := by
43         have h1 : c < 1 := h_c_lt_one
44         have h2 : c > 0 := by linarith
45         nlinarith [sq_pos_of_pos h2]
46         <=> nlinarith
47       have h_d2_lt_d : d^2 < d := by
48         have h1 : d < 1 := h_d_lt_one
49         have h2 : d > 0 := hd_pos
50         nlinarith [sq_pos_of_pos h2]
51         <=> nlinarith
52       have h_sum_sq_lt_one : a^2 + b^2 + c^2 + d^2 < 1 := by
53         nlinarith [h_a2_lt_a, h_b2_lt_b, h_c2_lt_c, h_d2_lt_d]
54         <=> linarith
55       have h_main : (a + 2*b + 3*c + 4*d) * (a^2 + b^2 + c^2 + d^2) < 1 := by
56         have h1 : 0 < a + 2 * b + 3 * c + 4 * d := by
57           nlinarith [hd_pos, hcb, hba, hdc, h_sum_eq_1]
58         have h2 : a ^ 2 + b ^ 2 + c ^ 2 + d ^ 2 < 1 := h_sum_sq_lt_one
59         nlinarith [h1, h2]
60         <=> nlinarith

```

```

69
70   exact h_main
71
72
73   theorem vars_are_in_0_1 (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b
74   ↪ ≤ a) (h1 : a + b + c + d = 1) :
75     (0 < a ∧ a < 1) ∧ (0 < b ∧ b < 1) ∧ (0 < c ∧ c < 1) ∧ (0 < d ∧ d < 1) := by
76       have h_a_pos : 0 < a := by
77         nlinarith [hdc, hcb, hba, hd0, h1]
78         <.> nlinarith
79
80       have h_a_lt_1 : a < 1 := by
81         have h2 : a < 1 := by
82           nlinarith [h1, h_a_pos, hba, hcb, hdc, hd0]
83         exact h2
84
85       have h_b_pos : 0 < b := by
86         nlinarith [hdc, hcb, hba, hd0, h1]
87
88       have h_b_lt_1 : b < 1 := by
89         have h2 : b < 1 := by
90           nlinarith [h1, h_a_pos, h_a_lt_1, hba, hcb, hdc, hd0]
91         exact h2
92
93       have h_c_pos : 0 < c := by
94         nlinarith [hdc, hcb, hba, hd0, h1]
95
96       have h_c_lt_1 : c < 1 := by
97         have h2 : c < 1 := by
98           nlinarith [h1, h_a_pos, h_a_lt_1, h_b_pos, h_b_lt_1, hba, hcb, hdc, hd0]
99         exact h2
100
101       have h_d_pos : 0 < d := by
102         exact hd0
103
104       have h_d_lt_1 : d < 1 := by
105         have h2 : d < 1 := by
106           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,
107             ↪ hba, hd0]
108         exact h2
109
110       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,
111         ↪ h_d_lt_1>>
112
113   theorem imo2020_q2 (a b c d : ℝ) (hd0 : 0 < d) (hdc : d ≤ c) (hcb : c ≤ b) (hba : b ≤ a)
114   ↪ (h1 : a + b + c + d = 1) :
115     (a + 2 * b + 3 * c + 4 * d) * (a ^ a * b ^ b * c ^ c * d ^ d) < 1 := by
116       -- strategy:
117       -- 1. apply weighted AM-GM inequality to prove a^a * b^b * c^c * d^d ≤ a^2 + b^2 + c^2 +
118       ↪ d^2
119       -- 2. use subgoal 'weighted_sum_times_sum_sq_lt_one' to get (a + 2*b + ...) * (a^2 + b^2
120       ↪ + ...) < 1
121       -- 3. combine both results to reach final conclusion
122
123       -- define S
124       let S := a^2 + b^2 + c^2 + d^2
125
126       -- step 1: apply weighted AM-GM inequality

```

```

122 -- we need to prove  $a^a * b^b * c^c * d^d \leq S$ 
123 have h_geom_mean_le_sum_sq :  $a^a * b^b * c^c * d^d \leq S$  := by
124   -- in order to use the subgoal 'geom_mean_le_arith_mean_weighted', we use Fin 4 as an
125   -- index type
126   let w : Fin 4 → ℝ := ![a, b, c, d]
127   let z : Fin 4 → ℝ := ![a, b, c, d]
128
129   -- check AM-GM prerequisite
130   have h_pos_conds : (0 < a) ∧ (0 < b) ∧ (0 < c) ∧ (0 < d) := by
131     have h_all := vars_are_in_0_1 a b c d hd0 hdc hcb hba h1
132     exact ⟨h_all.1.1, h_all.2.1.1, h_all.2.2.1.1, h_all.2.2.2.1⟩
133
134   -- 1. non-negative weights
135   have h_weights_nonneg : ∀ i, 0 ≤ w i := by
136     intro i; fin_cases i <;> simp [w] <;> linarith [h_pos_conds.1, h_pos_conds.2.1,
137       ↪ h_pos_conds.2.2.1, h_pos_conds.2.2.2]
138
139   -- 2. weights sum-up to 1
140   have h_weights_sum_1 : ∑ i, w i = 1 := by
141     simp [w, Fin.sum_univ_four, h1]
142
143   -- 3. non-negative values
144   have h_values_nonneg : ∀ i, 0 ≤ z i := by
145     intro i; fin_cases i <;> simp [z] <;> linarith [h_pos_conds.1, h_pos_conds.2.1,
146       ↪ h_pos_conds.2.2.1, h_pos_conds.2.2.2]
147
148   -- use the subgoal based on AM-GM
149   have h_am_gm := geom_mean_le_arith_mean_weighted (Finset.univ) w z (fun i _ ↪
150     ↪ h_weights_nonneg i) h_weights_sum_1 (fun i _ ↪ h_values_nonneg i)
151
152   -- transform AM-GM results to the form we want
153   -- `simp` will handle  $a^a \rightarrow a^2$ 
154   simp only [Fin.prod_univ_four, Fin.sum_univ_four, w, z, ← pow_two] at h_am_gm
155
156   -- it will replace 'S' to ' $a^2 + b^2 + c^2 + d^2$ '
157   unfold S
158   -- now the target fully matches 'h_am_gm'
159   exact h_am_gm
160
161   -- step 2: get results from key lemmas
162   have h_main_ineq :  $(a + 2 * b + 3 * c + 4 * d) * S < 1$  := by
163     exact weighted_sum_times_sum_sq_lt_one a b c d hd0 hdc hcb hba h1
164
165   -- step 3 & 4 & 5: assume final proof
166   calc
167     (a + 2*b + 3*c + 4*d) * (a^a * b^b * c^c * d^d)
168     -- first, use the results from step 1, we need to prove  $(a + 2*b + \dots)$  is positive
169     -- lemma 'vars_are_in_0_1' guarantees a,b,c,d > 0, thus their weighted sum also > 0
170     ≤ (a + 2*b + 3*c + 4*d) * S := by
171       apply mul_le_mul_of_nonneg_left h_geom_mean_le_sum_sq
172       have h_pos_conds := vars_are_in_0_1 a b c d hd0 hdc hcb hba h1
173       linarith [h_pos_conds.1.1, h_pos_conds.2.1.1, h_pos_conds.2.2.1.1,
174         ↪ h_pos_conds.2.2.2.1]
175     -- then, use the results from step 2 to finish proving
176     < 1 := h_main_ineq

```

## B.2 IMO 2019 P1

```

1  -- Solution to IMO 2019 P1 by DRP-IMO
2
3  import Mathlib
4  import Aesop
5
6  set_option maxHeartbeats 0
7
8  open BigOperators Real Nat Topology Rat
9
10 def solution_set (f :  $\mathbb{Z} \rightarrow \mathbb{Z}$ ) : Prop :=
11   ( $\forall x : \mathbb{Z}, f\ x = 0$ )  $\vee \exists c : \mathbb{Z}, \forall x : \mathbb{Z}, f\ x = 2 * x + c$ 
12
13 theorem linear_form_plus_f_f_x_implies_solutions (f :  $\mathbb{Z} \rightarrow \mathbb{Z}$ ) (c :  $\mathbb{Z}$ )
14   (h_f_form :  $\forall x, f\ x = c * x + f\ 0$ ) (h_f_f_x :  $\forall x, f\ (f\ x) = 2 * f\ x + f\ 0$ ) :
15   ( $\forall x, f\ x = 0$ )  $\vee (\forall x, f\ x = 2 * x + f\ 0)$  := by
16   have h_c_squared :  $c^2 = 2 * c$  := by
17     have h1 := h_f_f_x 1
18     have h2 := h_f_f_x 0
19     have h3 := h_f_f_x (-1)
20     have h4 := h_f_form 1
21     have h5 := h_f_form 0
22     have h6 := h_f_form (-1)
23     have h7 := h_f_form (f 1)
24     have h8 := h_f_form (f 0)
25     have h9 := h_f_form (f (-1))
26     have h10 := h_f_f_x (f 1)
27     have h11 := h_f_f_x (f 0)
28     have h12 := h_f_f_x (f (-1))
29     have h13 := h_f_form (c * 1 + f 0)
30     have h14 := h_f_form (c * 0 + f 0)
31     have h15 := h_f_form (c * (-1) + f 0)
32     have h16 := h_f_form (c * (f 1) + f 0)
33     have h17 := h_f_form (c * (f 0) + f 0)
34     have h18 := h_f_form (c * (f (-1)) + f 0)
35     ring_nf at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 ⊢
36     nlinarith [sq_nonneg (c - 2), sq_nonneg (c + 2), sq_nonneg (c - 1), sq_nonneg (c + 1)]
37
38   have h_c_cases :  $c = 0 \vee c = 2$  := by
39     have h1 :  $c^2 = 2 * c$  := h_c_squared
40     have h2 :  $c = 0 \vee c = 2$  := by
41       have h3 :  $c * (c - 2) = 0$  := by
42         linarith
43       have h4 :  $c = 0 \vee c - 2 = 0$  := by
44         apply eq_zero_or_eq_zero_of_mul_eq_zero h3
45       cases h4 with
46       | inl h4 =>
47         exact Or.inl h4
48       | inr h4 =>
49         have h5 :  $c = 2$  := by
50           omega
51         exact Or.inr h5
52     exact h2
53
54   have h_main : ( $\forall x, f\ x = 0$ )  $\vee (\forall x, f\ x = 2 * x + f\ 0)$  := by
55     cases h_c_cases with
56     | inl h_c_zero =>
57       -- Case  $c = 0$ 

```

```

58   have h_f_zero :  $\forall x, f x = f 0$  := by
59     intro x
60     have h1 := h_f_form x
61     simp [h_c_zero] at h1 ⊢
62     <;> linarith
63   have h_f_zero_zero :  $f 0 = 0$  := by
64     have h1 := h_f_f_x 0
65     have h2 := h_f_form 0
66     have h3 := h_f_form (f 0)
67     have h4 := h_f_f_x (f 0)
68     simp [h_f_zero] at h1 h2 h3 h4 ⊢
69     <;>
70     (try omega) <;>
71     (try
72       {
73         nlinarith [h_f_form 0, h_f_form 1, h_f_form (-1), h_f_form (f 0)]
74       } <;>
75     (try
76       {
77         cases' h_c_cases with h_c_zero h_c_two <;> simp_all [h_c_zero, h_c_two] <;>
78         (try omega) <;>
79         (try nlinarith) <;>
80         (try linarith)
81       } <;>
82     (try
83       {
84         aesop
85       })
86   have h_f_zero_all :  $\forall x, f x = 0$  := by
87     intro x
88     have h1 := h_f_zero x
89     have h2 := h_f_zero 0
90     have h3 := h_f_zero (-1)
91     have h4 := h_f_zero 1
92     simp [h_f_zero_zero] at h1 h2 h3 h4 ⊢
93     <;>
94     (try omega) <;>
95     (try nlinarith) <;>
96     (try aesop)
97     <;>
98     (try
99       {
100         simp_all [h_f_form, h_c_zero]
101         <;>
102         (try omega) <;>
103         (try nlinarith) <;>
104         (try aesop)
105       })
106   exact Or.inl h_f_zero_all
107 | inr h_c_two =>
108   -- Case c = 2
109   have h_f_form_two :  $\forall x, f x = 2 * x + f 0$  := by
110     intro x
111     have h1 := h_f_form x
112     simp [h_c_two] at h1 ⊢
113     <;> linarith
114   exact Or.inr h_f_form_two
115
116 exact h_main

```



```

117
118 theorem prop_cauchy_like (f :  $\mathbb{Z} \rightarrow \mathbb{Z}$ ) (h_f_all :  $\forall a b, f (2 * a) + 2 * (f b) = f (f (a +$ 
     $\rightarrow b)))$ 
119   (h_f_f_x :  $\forall x, f (f x) = 2 * f x + f 0$ ) (h_f_2x :  $\forall x, f (2 * x) = 2 * f x - f 0$ ) (x
     $\rightarrow y : \mathbb{Z}$ ) :
120   f (x + y) = f x + f y - f 0 := by
121   have h_main : f (x + y) = f x + f y - f 0 := by
122     have h1 := h_f_all (x + y) 0
123     have h2 := h_f_all x y
124     have h3 := h_f_all (x + y) y
125     have h4 := h_f_all x (x + y)
126     have h5 := h_f_2x (x + y)
127     have h6 := h_f_2x x
128     have h7 := h_f_2x y
129     have h8 := h_f_all 0 (x + y)
130     have h9 := h_f_all 0 x
131     have h10 := h_f_all 0 y
132     have h11 := h_f_f_x (x + y)
133     have h12 := h_f_f_x x
134     have h13 := h_f_f_x y
135     have h14 := h_f_all (2 * (x + y)) 0
136     have h15 := h_f_all (2 * x) 0
137     have h16 := h_f_all (2 * y) 0
138     have h17 := h_f_all x 0
139     have h18 := h_f_all y 0
140     have h19 := h_f_all (x + y) (x + y)
141     have h20 := h_f_all x x
142     have h21 := h_f_all y y
143     -- Simplify the expressions using the given conditions
144     simp [h_f_2x, mul_add, add_mul, mul_comm, mul_left_comm, mul_assoc] at h1 h2 h3 h4 h5
     $\rightarrow$  h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21  $\vdash$ 
145     <> ring_nf at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20
     $\rightarrow$  h21  $\vdash$ 
146     <> omega
147     exact h_main
148
149 theorem prop_f_f_x (f :  $\mathbb{Z} \rightarrow \mathbb{Z}$ ) (h_f_all :  $\forall a b, f (2 * a) + 2 * (f b) = f (f (a + b)))$ 
     $\rightarrow (x : \mathbb{Z}) :$ 
150   f (f x) = 2 * f x + f 0 := by
151   have h_main : f (f x) = 2 * f x + f 0 := by
152     have h1 := h_f_all x 0
153     have h2 := h_f_all 0 x
154     have h3 := h_f_all x x
155     have h4 := h_f_all (-x) x
156     have h5 := h_f_all x (-x)
157     have h6 := h_f_all 0 0
158     have h7 := h_f_all x (-2 * x)
159     have h8 := h_f_all (-x) (-x)
160     have h9 := h_f_all x 1
161     have h10 := h_f_all x (-1)
162     have h11 := h_f_all 1 x
163     have h12 := h_f_all (-1) x
164     have h13 := h_f_all 1 0
165     have h14 := h_f_all (-1) 0
166     have h15 := h_f_all 0 1
167     have h16 := h_f_all 0 (-1)
168     have h17 := h_f_all 1 1
169     have h18 := h_f_all (-1) (-1)
170     -- Simplify the equations to find a relationship between f(0) and f(f(0))

```

```

171     simp at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18
172     ring_nf at h1 h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 ⊢
173     -- Use linear arithmetic to solve for the desired result
174     omega
175     exact h_main
176
177 theorem prop_f_2x (f : ℤ → ℤ) (h_f_all : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b)))
178   (h_f_f_x : ∀ x, f (f x) = 2 * f x + f 0) (x : ℤ) :
179   f (2 * x) = 2 * f x - f 0 := by
180   have h1 : f (f (2 * x)) = 2 * f (2 * x) + f 0 := by
181     have h1 := h_f_f_x (2 * x)
182     -- Simplify the expression using the given condition h_f_f_x
183     simp at h1 ⊢
184     <|> linarith
185
186   have h2 : f (2 * x) + 2 * f x = f (f (2 * x)) := by
187     have h2 := h_f_all x x
188     -- Simplify the expression using the given condition h_f_all
189     ring_nf at h2 ⊢
190     <|> linarith
191
192   have h3 : f (2 * x) + 2 * f x = 2 * f (2 * x) + f 0 := by
193     have h3 : f (2 * x) + 2 * f x = f (f (2 * x)) := h2
194     rw [h3]
195     have h4 : f (f (2 * x)) = 2 * f (2 * x) + f 0 := h1
196     rw [h4]
197     <|> ring
198     <|> omega
199
200   have h4 : f (2 * x) = 2 * f x - f 0 := by
201     have h5 : f (2 * x) + 2 * f x = 2 * f (2 * x) + f 0 := h3
202     -- Rearrange the equation to isolate f(2 * x)
203     have h6 : f (2 * x) = 2 * f x - f 0 := by
204       -- Solve for f(2 * x) using linear arithmetic
205       linarith
206     exact h6
207
208   apply h4
209
210
211 theorem cauchy_implies_linear_form (f : ℤ → ℤ) (h_cauchy_like : ∀ x y, f (x + y) = f x +
212   ↪ f y - f 0) :
213   ∃ c : ℤ, ∀ x, f x = c * x + f 0 := by
214   have h_main : ∃ (c : ℤ), ∀ (x : ℤ), f x = c * x + f 0 := by
215     use f 1 - f 0
216     intro x
217     have h1 : ∀ n : ℤ, f n = (f 1 - f 0) * n + f 0 := by
218       intro n
219       induction n using Int.induction_on with
220       | hz =>
221         -- Base case: n = 0
222         simp [h_cauchy_like]
223         <|> ring_nf
224         <|> omega
225       | hp n ih =>
226         -- Inductive step: n = p + 1
227         have h2 := h_cauchy_like n 1
228         have h3 := h_cauchy_like 0 (n + 1)
229         have h4 := h_cauchy_like (n + 1) 0

```

```

229     have h5 := h_cauchy_like 1 0
230     have h6 := h_cauchy_like 0 1
231     simp at h2 h3 h4 h5 h6
232     simp [ih, add_mul, mul_add, mul_one, mul_neg, mul_zero, sub_eq_add_neg] at h2 h3
233     ↪ h4 h5 h6 ⊢
234     <;> ring_nf at *
235     <;> omega
236   | hn n ih =>
237     -- Inductive step: n = - (n + 1)
238     have h2 := h_cauchy_like (-n - 1) 1
239     have h3 := h_cauchy_like 0 (-n - 1)
240     have h4 := h_cauchy_like (-n - 1) 0
241     have h5 := h_cauchy_like 1 0
242     have h6 := h_cauchy_like 0 1
243     simp at h2 h3 h4 h5 h6
244     simp [ih, add_mul, mul_add, mul_one, mul_neg, mul_zero, sub_eq_add_neg] at h2 h3
245     ↪ h4 h5 h6 ⊢
246     <;> ring_nf at *
247     <;> omega
248   have h2 := h1 x
249   have h3 := h1 1
250   have h4 := h1 0
251   simp at h2 h3 h4 ⊢
252   <;> linarith
253   exact h_main
254
255 theorem step6_zero_function_is_solution (f : ℤ → ℤ) (h_zero : ∀ x, f x = 0) : (∀ a b, f
256   ↪ (2 * a) + 2 * (f b) = f (f (a + b))) := by
257   have h_main : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b)) := by
258     intro a b
259     have h1 : f (2 * a) = 0 := by
260       rw [h_zero]
261       <;> simp [h_zero]
262     have h2 : f b = 0 := by
263       rw [h_zero]
264       <;> simp [h_zero]
265     have h3 : f (a + b) = 0 := by
266       rw [h_zero]
267       <;> simp [h_zero]
268     have h4 : f (f (a + b)) = 0 := by
269       rw [h_zero]
270       <;> simp [h_zero]
271     -- Simplify the LHS and RHS using the above equalities
272     simp [h1, h2, h3, h4, h_zero]
273     <;> linarith
274   exact h_main
275
276 theorem step7_linear_function_is_solution (f : ℤ → ℤ) (c : ℤ) (h_lin : ∀ x, f x = 2 * x
277   ↪ + c) : (∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b))) := by
278   have h_main : ∀ a b, f (2 * a) + 2 * (f b) = f (f (a + b)) := by
279     intro a b
280     have h1 : f (2 * a) = 2 * (2 * a) + c := by
281       rw [h_lin]
282       <;> ring
283     have h2 : f b = 2 * b + c := by
284       rw [h_lin]
285       <;> ring
286     have h3 : f (f (a + b)) = f (2 * (a + b) + c) := by
287       rw [h_lin]
288       <;> ring
289     have h4 : f (a + b) = 2 * (a + b) + c := by
290       rw [h_lin]
291       <;> ring

```

```

284     rw [h_lin]
285     <|> ring
286     rw [h4]
287     <|> ring
288     have h4 : f (f (a + b)) = 2 * (2 * (a + b) + c) + c := by
289       rw [h3]
290       rw [h_lin]
291       <|> ring
292     have h5 : f (2 * a) + 2 * (f b) = (2 * (2 * a) + c) + 2 * (2 * b + c) := by
293       rw [h1, h2]
294       <|> ring
295     have h6 : f (2 * a) + 2 * (f b) = 4 * a + 4 * b + 3 * c := by
296       linarith
297     have h7 : f (f (a + b)) = 4 * a + 4 * b + 3 * c := by
298       linarith
299     linarith
300   exact h_main
301
302   theorem imo2019_p1
303     (f : ℤ → ℤ) :
304       (∀ a b : ℤ, f (2 * a) + 2 * f b = f (f (a + b))) ↔ solution_set f := by
305     constructor
306     · intro h_fe
307       have h_ff : ∀ x, f (f x) = 2 * f x + f 0 :=
308         prop_f_f_x f h_fe
309       have h_f2 : ∀ x, f (2 * x) = 2 * f x - f 0 :=
310         prop_f_2x f h_fe h_ff
311       have h_add : ∀ x y, f (x + y) = f x + f y - f 0 :=
312         prop_cauchy_like f h_fe h_ff h_f2
313       rcases cauchy_implies_linear_form f h_add with ⟨c, h_lin0⟩
314       have h_split :
315         (∀ x, f x = 0) ∨ (∀ x, f x = 2 * x + f 0) :=
316         linear_form_plus_f_f_x_implies_solutions f c h_lin0 h_ff
317       cases h_split with
318       | inl h0 =>
319         exact Or.inl h0
320       | inr h2 =>
321         exact Or.inr ⟨f 0, h2⟩
322     · intro h_sol
323       cases h_sol with
324       | inl h0 =>
325         exact step6_zero_function_is_solution f h0
326       | inr h_exists =>
327         rcases h_exists with ⟨c, h_lin⟩
328         exact step7_linear_function_is_solution f c h_lin

```

### B.3 IMO 2011 P3

```

1  -- Solution to IMO 2011 P3 by DRP-IMO
2
3  import Mathlib
4  import Aesop
5
6  set_option maxHeartbeats 0
7
8  open BigOperators Real Nat Topology Rat
9

```

```

10
11
12 theorem imo2011_p3_lemma1_f_neg_le_self (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x +
    ↪ f (f x)) :
13   ∀ x, f x < 0 → f x ≤ x := by
14   have h_main : ∀ (x : ℝ), f x < 0 → f x ≤ x := by
15     intro x hx
16     have h1 : f x ^ 2 - x * f x ≥ 0 := by
17       have h2 := hf x (f x - x)
18       have h3 := hf (f x) (x - f x)
19       have h4 := hf x 0
20       have h5 := hf 0 x
21       have h6 := hf x x
22       have h7 := hf x (-x)
23       have h8 := hf (-x) x
24       have h9 := hf 0 0
25       have h10 := hf x 1
26       have h11 := hf 1 x
27       have h12 := hf x (-1)
28       have h13 := hf (-1) x
29       have h14 := hf x (f x)
30       have h15 := hf (f x) x
31       have h16 := hf x (-f x)
32       have h17 := hf (-f x) x
33       have h18 := hf x (x + f x)
34       have h19 := hf (x + f x) x
35       have h20 := hf x (-x)
36       have h21 := hf (-x) x
37       have h22 := hf x (x - f x)
38       have h23 := hf (x - f x) x
39       have h24 := hf x (f x + x)
40       have h25 := hf (f x + x) x
41       have h26 := hf x (2 * f x)
42       have h27 := hf (2 * f x) x
43       have h28 := hf x (-2 * f x)
44       have h29 := hf (-2 * f x) x
45     -- Normalize the expressions to simplify the inequalities
46     ring_nf at h2 h3 h4 h5 h6 h7 h8 h9 h10 h11 h12 h13 h14 h15 h16 h17 h18 h19 h20 h21
47     ↪ h22 h23 h24 h25 h26 h27 h28 h29 ⊢
48     -- Use linear arithmetic to prove the inequality
49     nlinarith [sq_nonneg (f x - x), sq_nonneg (f x + x), sq_nonneg (f x - 2 * x),
50       ↪ sq_nonneg (f x + 2 * x),
51       sq_nonneg (2 * f x - x), sq_nonneg (2 * f x + x)]
52   have h3 : f x ≤ x := by
53     by_contra h
54     have h4 : f x > x := by linarith
55     have h5 : f x ^ 2 - x * f x < 0 := by
56       nlinarith [hx, h4]
57     nlinarith
58     exact h3
59   exact h_main
60
61 /--Let  $(f : \mathbb{R} \rightarrow \mathbb{R})$  be a function such that for all real numbers  $(x)$ 
62 ↪ 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.-/
63 theorem imo2011_p3_st1 (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x)) :
64   ∀ x, f x ≤ f (f x) := by
65   have h_main : ∀ (x : ℝ), f x ≤ f (f x) := by

```

```

64     intro x
65     have h1 := hf x 0
66     -- Simplify the inequality by substituting y = 0
67     simp at h1
68     -- Use the simplified inequality to conclude the proof
69     linarith
70     exact h_main
71
72
73
74 /--Consider a function  $f : \mathbb{R} \rightarrow \mathbb{R}$  that satisfies the condition:
75  $\rightarrow$  for all real numbers  $x$  and  $y$ ,  $|f(x+y)| \leq y |f(x) + f(f(x))|$ . Prove that for all real numbers  $x$ ,  $|f(x)| \leq 0$ .- /
76 theorem aux_f_nonpositive (f :  $\mathbb{R} \rightarrow \mathbb{R}$ ) (hf :  $\forall x y, f(x+y) \leq y * f x + f(f x)$ ) :  $\forall x,$ 
77  $\rightarrow f x \leq 0 :=$  by
78 have h_main :  $\forall x, f x \leq 0 :=$  by
79     intro x
80     by_contra h
81     have h1 : f x > 0 := by linarith
82     have h2 := hf x (-x)
83     have h3 := hf 0 (f x)
84     have h4 := hf x 0
85     have h5 := hf (-x) x
86     have h6 := hf (-x) (-x)
87     have h7 := hf x (f x)
88     have h8 := hf (f x) (-f x)
89     have h9 := hf (f x) 0
90     have h10 := hf 0 (-f x)
91     have h11 := hf x (2 * x)
92     have h12 := hf x (-2 * x)
93     have h13 := hf (2 * x) (-x)
94     have h14 := hf (2 * x) x
95     have h15 := hf (-2 * x) x
96     have h16 := hf (-2 * x) (-x)
97     have h17 := hf (f x) x
98     have h18 := hf (f x) (-x)
99     have h19 := hf x (f x)
100    have h20 := hf (-x) (f x)
101    have h21 := hf x (-f x)
102    have h22 := hf (-x) (-f x)
103    have h23 := hf (2 * f x) (-f x)
104    have h24 := hf (-2 * f x) (-f x)
105    have h25 := hf (2 * f x) (f x)
106    have h26 := hf (-2 * f x) (f x)
107    have h27 := hf (f x) (2 * f x)
108    have h28 := hf (f x) (-2 * f x)
109    have h29 := hf x (x)
110    have h30 := hf x (-x)
111    have h31 := hf 0 (2 * f x)
112    have h32 := hf 0 (-2 * f x)
113    have h33 := hf (2 * f x) 0
114    have h34 := hf (-2 * f x) 0
115    have h35 := hf (f x) (f x)
116    have h36 := hf (-f x) (f x)
117    have h37 := hf (f x) (-f x)
118    have h38 := hf (-f x) (-f x)
119    norm_num at *
120    <;>
    (try nlinarith) <;>

```



```

121 (try linarith) <=>
122 (try nlinarith [h1, h2, h3, h4, h5, h6, h7, h8, h9, h10, h11, h12, h13, h14, h15, h16, h17,
123   ↪ h18, h19, h20, h21, h22, h23, h24, h25, h26, h27, h28, h2
124   9, h30, h31, h32, h33, h34, h35, h36, h37, h38]) <=>
125 (try
126   nlinarith [hf 0 0, hf x 0, hf 0 x, hf x (-x), hf (-x) x, hf (x + x) (-x), hf (-x) (x +
127     ↪ x), hf (x - x) (x + x), hf (x + x) (x - x)]) <=>
128 (try
129   nlinarith [hf 0 0, hf x 0, hf 0 x, hf x (-x), hf (-x) x, hf (x + x) (-x), hf (-x) (x +
130     ↪ x), hf (x - x) (x + x), hf (x + x) (x - x)])
131 <=>
132 nlinarith
133 exact h_main
134
135
136
137 theorem lemma_final_implication (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x))
138   (h_f_at_0_is_0 : f 0 = 0) (h_f_non_positive : ∀ x, f x ≤ 0) : ∀ x ≤ 0, f x = 0 := by
139   have h_main : ∀ (x : ℝ), x ≤ 0 → f x = 0 := by
140     intro x hx
141     have h1 : f x = 0 := by
142       by_cases hx0 : x = 0
143       · -- If x = 0, then f(0) = 0 by hypothesis
144         simp [hx0, h_f_at_0_is_0]
145       · -- If x ≠ 0, then x < 0
146         have hx1 : x < 0 := by
147           cases' lt_or_gt_of_ne hx0 with h h
148           · linarith
149           · exfalso
150           linarith
151         -- Use the given inequality with y = x and y = -x to derive the desired result
152         have h2 := hf x x
153         have h3 := hf (-x) x
154         have h4 := hf x (-x)
155         have h5 := hf 0 x
156         have h6 := hf x 0
157         have h7 := hf 0 (-x)
158         have h8 := hf (-x) 0
159         -- Simplify the inequalities using the given conditions
160         norm_num [h_f_at_0_is_0] at h2 h3 h4 h5 h6 h7 h8 ⊢
161         nlinarith [h_f_non_positive x, h_f_non_positive (-x), h_f_non_positive (f x),
162           h_f_non_positive (x + x), h_f_non_positive (x - x), h_f_non_positive 0]
163     exact h1
164   exact h_main
165
166 /--Let  $f : \mathbb{R} \rightarrow \mathbb{R}$  be a function satisfying the inequality  $f(x + y) \leq y \cdot f(x) + f(f(x))$  for all real numbers  $x$ 
167   and  $f(0) = 0$ . 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
168   real numbers  $y$ .-/>
169 theorem lemma_properties_if_f_has_zero (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f
170   ↪ (f x))
171   (h_f0_eq_c : f 0 = c) (hx0 : ∃ x0, f x0 = 0) :
172   c ≥ 0 ∧ f c = c ∧ (∀ y, f y ≤ c) ∧ (∀ y, f y ≤ c * y + c) := by

```

```

172 have h_c_ge_zero : c ≥ 0 := by
173   obtain ⟨x₀, hx₀⟩ := hx₀
174   have h1 := hf x₀ (-x₀)
175   have h2 := hf 0 (-x₀)
176   have h3 := hf x₀ 0
177   have h4 := hf 0 0
178   have h5 := hf x₀ (f x₀)
179   have h6 := hf 0 (f 0)
180   have h7 := hf x₀ (-f x₀)
181   have h8 := hf 0 (-f 0)
182   norm_num [h_f0_eq_c, hx₀] at *
183   <|>
184   (try linarith) <|>
185   (try nlinarith) <|>
186   (try simp_all [h_f0_eq_c, hx₀]) <|>
187   (try linarith) <|>
188   (try nlinarith)
189   <|>
190   (try
191     nlinarith [sq_nonneg (f x₀), sq_nonneg (f 0), sq_nonneg (x₀ + 0), sq_nonneg (x₀ - 0),
192       ↪ sq_nonneg (f x₀ + f 0), sq_nonneg (f x₀ - f 0)])
193   <|>
194   (try
195     nlinarith [sq_nonneg (f x₀), sq_nonneg (f 0), sq_nonneg (x₀ + 0), sq_nonneg (x₀ - 0),
196       ↪ sq_nonneg (f x₀ + f 0), sq_nonneg (f x₀ - f 0)])
197
198 have h_f_c_eq_c : f c = c := by
199   obtain ⟨x₀, hx₀⟩ := hx₀
200   have h1 := hf x₀ (c - x₀)
201   have h2 := hf 0 (c)
202   have h3 := hf c (-c)
203   have h4 := hf x₀ 0
204   have h5 := hf 0 0
205   have h6 := hf x₀ (f x₀)
206   have h7 := hf 0 (f 0)
207   have h8 := hf x₀ (-x₀)
208   have h9 := hf 0 (-x₀)
209   simp [h_f0_eq_c, hx₀] at h1 h2 h3 h4 h5 h6 h7 h8 h9 ⊢
210   <|>
211   (try ring_nf at * <|> nlinarith) <|>
212   (try
213     {
214       nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀)]
215     } <|>
216     (try
217       {
218         nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀), sq_nonneg (f
219           ↪ c)]
220       }
221     )
222   )
223   <|>
224   (try
225     {
226       nlinarith [sq_nonneg (f x₀), sq_nonneg (c - x₀), sq_nonneg (c + x₀), sq_nonneg (f
227         ↪ c), sq_nonneg (f 0), sq_nonneg (c - f x₀)]

```

```

227     })
228     <;>
229     (try
230     {
231         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 -
232     c)]
233     })
234
235     have h_f_le_c : ∀ y, f y ≤ c := by
236         intro y
237         have h1 := hf y (-y)
238         have h2 := hf y (c - y)
239         have h3 := hf 0 (y)
240         have h4 := hf c (-c)
241         have h5 := hf y 0
242         have h6 := hf 0 0
243         have h7 := hf c 0
244         have h8 := hf 0 c
245         have h9 := hf y (f y)
246         have h10 := hf 0 (f 0)
247         have h11 := hf y (-f y)
248         have h12 := hf 0 (-f 0)
249         have h13 := hf c (y - c)
250         have h14 := hf 0 (y - c)
251         have h15 := hf (y - c) c
252         have h16 := hf (y - c) 0
253         have h17 := hf (y - c) (f (y - c))
254         have h18 := hf (y - c) (-f (y - c))
255         norm_num [h_f0_eq_c, h_f_c_eq_c] at *
256         <;>
257         (try linarith) <;>
258         (try nlinarith) <;>
259         (try
260         {
261             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)]
262         }) <;>
263         (try
264         {
265             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]
266         }) <;>
267         (try
268         {
269             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
270     y)]
271         }) <;>
272         (try
273         {
274             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
275     y - c)]
276         })
277
278     have h_f_le_cy_add_c : ∀ y, f y ≤ c * y + c := by
279         intro y
280         have h1 := h_f_le_c y

```

```

281   have h2 := h_f_le_c 0
282   have h3 := hf 0 y
283   have h4 := hf y 0
284   have h5 := hf y (-y)
285   have h6 := hf 0 (-y)
286   have h7 := hf y (c - y)
287   have h8 := hf 0 (c)
288   have h9 := hf c (-c)
289   have h10 := hf y 0
290   have h11 := hf 0 0
291   have h12 := hf y (f y)
292   have h13 := hf 0 (f 0)
293   have h14 := hf y (-f y)
294   have h15 := hf 0 (-f 0)
295   have h16 := hf c (y - c)
296   have h17 := hf 0 (y - c)
297   have h18 := hf (y - c) c
298   have h19 := hf (y - c) 0
299   have h20 := hf (y - c) (f (y - c))
300   have h21 := hf (y - c) (-f (y - c))
301   norm_num [h_f0_eq_c, h_f_c_eq_c] at *
302   <.>
303   (try linarith) <.>
304   (try nlinarith) <.>
305   (try
306     {
307       nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y),
308         ↪ sq_nonneg (f y + c - c * y)]
309     }) <.>
310   (try
311     {
312       nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y),
313         ↪ sq_nonneg (f y + c - c * y), sq_nonneg (f y - c * y)]
314     }) <.>
315   (try
316     {
317       nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y),
318         ↪ sq_nonneg (f y + c - c * y), sq_nonneg (f y - c * y), sq_nonneg
319         (f y - c)]
320     })
321   <.>
322   (try
323     {
324       cases' le_total 0 y with hy hy <.>
325       cases' le_total 0 (f y - c) with h h <.>
326       cases' le_total 0 (c - y) with h' h' <.>
327       nlinarith [h_c_ge_zero, sq_nonneg (f y - c), sq_nonneg (y), sq_nonneg (c - y),
328         ↪ sq_nonneg (f y + c - c * y), sq_nonneg (f y - c * y)]
329     })
330   <.>
331   nlinarith
332
333   exact ⟨h_c_ge_zero, h_f_c_eq_c, h_f_le_c, h_f_le_cy_add_c⟩
334
335 theorem imo2011_p3 (f : ℝ → ℝ) (hf : ∀ x y, f (x + y) ≤ y * f x + f (f x)) : ∀ x ≤ 0,
336   ↪ f x = 0 := by
337   -- Step 1: Prove that f is non-positive everywhere.
338   have h_nonpos : ∀ x, f x ≤ 0 := aux_f_nonpositive f hf
339

```

```

335 -- Step 2: Prove that there must exist some  $x_0$  such that  $f(x_0) = 0$ .
336 -- We prove this by contradiction. Assume  $f(x)$  is never zero.
337 have h_exists_zero :  $\exists x_0, f x_0 = 0$  := by
338   by_contra h_no_zero
339   -- The hypothesis from `by_contra` is `h_no_zero :  $\neg(\exists x_0, f x_0 = 0)$ `.
340   -- We use `push_neg` to convert it into a more usable form.
341   push_neg at h_no_zero
342   -- Now, `h_no_zero :  $\forall (x : \mathbb{R}), f x \neq 0$ `.
343
344   -- This, combined with `h_nonpos`, implies  $f(x) < 0$  for all  $x$ .
345   have h_always_neg :  $\forall x, f x < 0$  := fun x => (h_nonpos x).lt_of_ne (h_no_zero x)
346
347   -- From this, we can deduce  $f(0) = f(f(0))$ .
348   have h_f0_eq_ff0 :  $f 0 = f (f 0)$  := by
349     have h_ff0_neg :  $f (f 0) < 0$  := h_always_neg (f 0)
350     have hle :  $f (f 0) \leq f 0$  := imo2011_p3_lemma1_f_neg_le_self f hf (f 0) h_ff0_neg
351     have hge :  $f 0 \leq f (f 0)$  := imo2011_p3_st1 f hf 0
352     linarith
353
354   -- Now, we use the main inequality to derive a contradiction.
355   -- Let  $x = f(0)$  and  $y = -f(0)$ .
356   specialize hf (f 0) (-f 0)
357
358   -- Define a local lemma for ` $a + -a = 0$ ` to ensure it's available.
359   have add_neg_self_local :  $f 0 + -f 0 = 0$  := by ring
360
361   -- Rewrite the inequality step-by-step to derive the contradiction.
362   rw [add_neg_self_local] at hf
363   rw [← h_f0_eq_ff0] at hf
364   rw [← h_f0_eq_ff0] at hf
365
366   -- The inequality is now  $f 0 \leq -(f 0)^2 + f 0$ , which implies  $0 \leq -(f 0)^2$ .
367   have h_contr :  $0 \leq -(f 0)^2$  := by linarith [hf]
368
369   -- This is a contradiction because  $f(0) < 0$ , so  $-(f 0)^2 < 0$ .
370   have h_f0_neg :  $f 0 < 0$  := h_always_neg 0
371   have h_sq_pos :  $0 < (f 0)^2$  := sq_pos_of_ne_zero (ne_of_lt h_f0_neg)
372   linarith
373
374 -- Step 3: Use the existence of a zero to prove  $f(0) = 0$ .
375 obtain ⟨x₀, hx₀⟩ := h_exists_zero
376 have h_f0_eq_0 :  $f 0 = 0$  := by
377   -- A lemma gives properties of  $f$  if it has a zero. One is  $f(0) \geq 0$ .
378   have h_props := lemma_properties_if_f_has_zero f hf rfl ⟨x₀, hx₀⟩
379   have h_f0_nonneg :  $f 0 \geq 0$  := h_props.1
380   -- Combining  $f(0) \geq 0$  with  $f(0) \leq 0$  (from  $h_{\text{nonpos}}$ ) gives  $f(0) = 0$ .
381   linarith [h_nonpos 0]
382
383 -- Step 4: Now that we have  $f(x) \leq 0$  and  $f(0) = 0$ , apply the final lemma.
384 exact lemma_final_implication f hf h_f0_eq_0 h_nonpos

```

## B.4 IMO 2005 P3

```

1 -- Solution to IMO 2005 P3 by DRP-IMO
2
3 import Mathlib
4 import Aesop

```

```

5
6 set_option maxHeartbeats 0
7
8 open BigOperators Real Nat Topology Rat
9
10
11 theorem inequality_part1_nonnegative (x y z : ℝ) (hx : x > 0) (hy : y > 0) (hz : z > 0)
12   → (h : x * y * z ≥ 1) :
13     (x*x - 1/x + y*y - 1/y + z*z - 1/z) / (x*x + y*y + z*z) ≥ 0 := by
14       have h_main : x*x + y*y + z*z - (1/x + 1/y + 1/z) ≥ 0 := by
15         have h1 : 0 < x * y := by positivity
16         have h2 : 0 < x * z := by positivity
17         have h3 : 0 < y * z := by positivity
18         have h4 : 0 < x * y * z := by positivity
19         have h5 : 0 < x * y * z * x := by positivity
20         have h6 : 0 < x * y * z * y := by positivity
21         have h7 : 0 < x * y * z * z := by positivity
22         field_simp [hx.ne', hy.ne', hz.ne']
23         rw [le_div_iff0 (by positivity)]
24         -- Use nlinarith to prove the inequality
25         nlinarith [sq_nonneg (x - y), sq_nonneg (x - z), sq_nonneg (y - z),
26           sq_nonneg (x * y - 1), sq_nonneg (x * z - 1), sq_nonneg (y * z - 1),
27           mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x - y)),
28           mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x - z)),
29           mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (y - z)),
30           mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x * y - x * z)),
31           mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x * y - y * z)),
32           mul_nonneg (sub_nonneg.mpr h) (sq_nonneg (x * z - y * z))]
33
34       have h_final : (x*x - 1/x + y*y - 1/y + z*z - 1/z) / (x*x + y*y + z*z) ≥ 0 := by
35         have h1 : x * x + y * y + z * z - (1 / x + 1 / y + 1 / z) ≥ 0 := h_main
36         have h2 : x * x + y * y + z * z > 0 := by positivity
37         have h3 : (x * x - 1 / x + y * y - 1 / y + z * z - 1 / z) / (x * x + y * y + z * z) ≥
38           → 0 := by
39           have h4 : x * x - 1 / x + y * y - 1 / y + z * z - 1 / z = (x * x + y * y + z * z) -
40             → (1 / x + 1 / y + 1 / z) := by
41             ring
42             rw [h4]
43             have h5 : ((x * x + y * y + z * z) - (1 / x + 1 / y + 1 / z)) / (x * x + y * y + z *
44               → z) ≥ 0 := by
45               apply div_nonneg
46               · linarith
47               · linarith
48             exact h5
49             exact h3
50
51       exact h_final
52
53 theorem inequality_part2_nonnegative (x y z : ℝ) (hx : x > 0) (hy : y > 0) (hz : z > 0) :
54   ((x^5 - x^2)/(x^5 + y^2 + z^2) - (x*x - 1/x)/(x*x + y*y + z*z)) +
55   ((y^5 - y^2)/(y^5 + z^2 + x^2) - (y*y - 1/y)/(y*y + z^2 + x*x)) +
56   ((z^5 - z^2)/(z^5 + x^2 + y^2) - (z*z - 1/z)/(z*z + x*x + y*y)) ≥ 0 := by
57     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 -
58       → y^2)/(y^5 + z^2 + x^2) - (y*y - 1/y)/(y*y + z^2 + x*x)) + ((z
59       → ^5 - z^2)/(z^5 + x^2 + y^2) - (z*z - 1/z)/(z*z + x*x + y*y)) ≥ 0 := by
60       have h1 : (x^5 - x^2)/(x^5 + y^2 + z^2) - (x*x - 1/x)/(x*x + y*y + z*z) ≥ 0 := by
61         have h10 : 0 < x^5 + y^2 + z^2 := by positivity
62         have h11 : 0 < x*x + y*y + z*z := by positivity

```



```

59   have h12 : 0 < x^5 := by positivity
60   have h13 : 0 < x^3 := by positivity
61   have h14 : 0 < x^2 := by positivity
62   have h15 : 0 < x := by positivity
63   field_simp
64   rw [le_div_iff0 (by positivity), ← sub_nonneg]
65   ring_nf
66   nlinarith [sq_nonneg (x^3 - x), sq_nonneg (x^2 - 1), sq_nonneg (x - 1),
67     mul_nonneg hx.le (sq_nonneg (x^2 - 1)), mul_nonneg hx.le (sq_nonneg (x^3 - x)),
68     mul_nonneg hx.le (sq_nonneg (x^2 - x)), mul_nonneg hx.le (sq_nonneg (x^3 - 1)),
69     mul_nonneg (sq_nonneg (x - 1)) (sq_nonneg (x + 1)), mul_nonneg hx.le (sq_nonneg
    ↪ (x^2 - 2 * x + 1))]
70   have h2 : (y^5 - y^2)/(y^5 + z^2 + x^2) - (y*y - 1/y)/(y*y + z^2 + x*x) ≥ 0 := by
71     have h20 : 0 < y^5 + z^2 + x^2 := by positivity
72     have h21 : 0 < y*y + z^2 + x*x := by positivity
73     have h22 : 0 < y^5 := by positivity
74     have h23 : 0 < y^3 := by positivity
75     have h24 : 0 < y^2 := by positivity
76     have h25 : 0 < y := by positivity
77     field_simp
78     rw [le_div_iff0 (by positivity), ← sub_nonneg]
79     ring_nf
80     nlinarith [sq_nonneg (y^3 - y), sq_nonneg (y^2 - 1), sq_nonneg (y - 1),
81       mul_nonneg hy.le (sq_nonneg (y^2 - 1)), mul_nonneg hy.le (sq_nonneg (y^3 - y)),
82       mul_nonneg hy.le (sq_nonneg (y^2 - y)), mul_nonneg hy.le (sq_nonneg (y^3 - 1)),
83       mul_nonneg (sq_nonneg (y - 1)) (sq_nonneg (y + 1)), mul_nonneg hy.le (sq_nonneg
    ↪ (y^2 - 2 * y + 1))]
84   have h3 : (z^5 - z^2)/(z^5 + x^2 + y^2) - (z*z - 1/z)/(z*z + x*x + y*y) ≥ 0 := by
85     have h30 : 0 < z^5 + x^2 + y^2 := by positivity
86     have h31 : 0 < z*z + x*x + y*y := by positivity
87     have h32 : 0 < z^5 := by positivity
88     have h33 : 0 < z^3 := by positivity
89     have h34 : 0 < z^2 := by positivity
90     have h35 : 0 < z := by positivity
91     field_simp
92     rw [le_div_iff0 (by positivity), ← sub_nonneg]
93     ring_nf
94     nlinarith [sq_nonneg (z^3 - z), sq_nonneg (z^2 - 1), sq_nonneg (z - 1),
95       mul_nonneg hz.le (sq_nonneg (z^2 - 1)), mul_nonneg hz.le (sq_nonneg (z^3 - z)),
96       mul_nonneg hz.le (sq_nonneg (z^2 - z)), mul_nonneg hz.le (sq_nonneg (z^3 - 1)),
97       mul_nonneg (sq_nonneg (z - 1)) (sq_nonneg (z + 1)), mul_nonneg hz.le (sq_nonneg
    ↪ (z^2 - 2 * z + 1))]
98   linarith
99   exact h_main
100
101
102 -- The main theorem
103 theorem imo2005_p3 (x y z : ℝ) (hx : x > 0) (hy : y > 0) (hz : z > 0) (h_prod_ge_1 : x * y
    ↪ * z ≥ 1) :
104   (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
105     -- Define S_part1 (LHS of inequality_part1_nonnegative)
106     let S_part1 := (x^2 - 1/x + y^2 - 1/y + z^2 - 1/z) / (x^2 + y^2 + z^2)
107
108     -- Define S_part2 (LHS of inequality_part2_nonnegative)
109     let S_part2 :=
110       ((x^5 - x^2)/(x^5 + y^2 + z^2) - (x^2 - 1/x)/(x^2 + y^2 + z^2)) +
111       ((y^5 - y^2)/(y^5 + z^2 + x^2) - (y^2 - 1/y)/(y^2 + z^2 + x^2)) +
112       ((z^5 - z^2)/(z^5 + x^2 + y^2) - (z^2 - 1/z)/(z^2 + x^2 + y^2))

```

```

113
114 -- Prove  $S_{\text{part1}} \geq 0$  using inequality_part1_nonnegative
115 have h_S_part1_nonneg :  $S_{\text{part1}} \geq 0$  := by
116   apply inequality_part1_nonnegative <|> assumption
117
118 -- Prove  $S_{\text{part2}} \geq 0$  using inequality_part2_nonnegative
119 have h_S_part2_nonneg :  $S_{\text{part2}} \geq 0$  := by
120   apply inequality_part2_nonnegative <|> assumption
121
122 -- Prove that the original LHS is equal to  $S_{\text{part1}} + S_{\text{part2}}$ 
123 -- We'll prove it as a separate fact (have) and then use it.
124 have h_LHS_eq_sum :
125    $(x^5 - x^2)/(x^5 + y^2 + z^2) + (y^5 - y^2)/(y^5 + z^2 + x^2) + (z^5 - z^2)/(z^5 + x^2 + y^2) =$ 
126    $S_{\text{part2}} + S_{\text{part1}}$  := by
127   -- Expand the definitions of  $S_{\text{part1}}$  and  $S_{\text{part2}}$ 
128   unfold S_part1 S_part2
129   -- Normalize denominators which are permutations of each other
130   have h_denom_y :  $y^2 + z^2 + x^2 = x^2 + y^2 + z^2$  := by ac_rfl
131   have h_denom_z :  $z^2 + x^2 + y^2 = x^2 + y^2 + z^2$  := by ac_rfl
132   rw [h_denom_y, h_denom_z]
133   -- The rest is a pure algebraic identity, which `ring` can solve.
134   -- It correctly rearranges terms like  $(a-b)+(c-d)+(e-f) + (b+d+f)/k = a+c+e$ 
135   -- after combining the fractions for  $S_{\text{part1}}$ 
136   ring
137
138 -- Rewrite the goal using the equality we just proved
139 rw [h_LHS_eq_sum]
140
141 -- The goal is now  $S_{\text{part2}} + S_{\text{part1}} \geq 0$ , which follows from the two parts being
142    $\hookrightarrow$  non-negative.
exact add_nonneg h_S_part2_nonneg h_S_part1_nonneg

```

## B.5 IMO 2000 P2

```

1 -- Solution to IMO 2000 P2 by DRP-IMO
2
3 import Mathlib
4 import Aesop
5
6 set_option maxHeartbeats 0
7
8 open BigOperators Real Nat Topology Rat
9
10 /--Given positive real numbers  $| (a)|$ ,  $| (b)|$ , and  $| (c)|$  such that  $| (a) \text{ times } b$ 
11    $\hookrightarrow | \text{ times } c = 1 |$ , prove that there exist positive real numbers  $| (x$ 
12    $|)$ ,  $| (y)|$ , and  $| (z)|$  such that  $| (a = \frac{x}{y})|$ ,  $| (b = \frac{y}{z})|$ , and  $| (c$ 
13    $\hookrightarrow = \frac{z}{x})|$ .- /
14 theorem imo2000_p2_existence_of_xyz (a b c : ℝ) (ha : 0 < a) (hb : 0 < b) (hc : 0 < c)
15    $\hookrightarrow$  (habc : a * b * c = 1) :
16    $\exists x y z : \mathbb{R}, 0 < x \wedge 0 < y \wedge 0 < z \wedge a = x/y \wedge b = y/z \wedge c = z/x$  := by
17   have h_main :  $\exists (x y z : \mathbb{R}), 0 < x \wedge 0 < y \wedge 0 < z \wedge a = x/y \wedge b = y/z \wedge c = z/x$  :=
18      $\hookrightarrow$  by
19       refine' ⟨a, 1, 1 / b, _, _, _, _, _⟩
20       · -- Prove that  $a > 0$ 
21         linarith
22       · -- Prove that  $1 > 0$ 

```

```

19     norm_num
20     · -- Prove that  $1 / b > 0$ 
21     exact div_pos zero_lt_one hb
22     · -- Prove that  $a = a / 1$ 
23     field_simp
24     · -- Prove that  $b = 1 / (1 / b)$ 
25     field_simp
26     <.>
27     nlinarith
28     · -- Prove that  $c = (1 / b) / a$ 
29     have h1 :  $c = 1 / (a * b) := by$ 
30       have h2 :  $a * b * c = 1 := habc$ 
31       have h3 :  $c = 1 / (a * b) := by$ 
32         have h4 :  $a * b \neq 0 := by$  positivity
33         field_simp [h4] at h2 ⊢
34         nlinarith
35     exact h3
36     have h2 :  $(1 / b : \mathbb{R}) / a = 1 / (a * b) := by$ 
37       field_simp
38       <.> ring
39       <.> field_simp [ha.ne', hb.ne']
40       <.> nlinarith
41     rw [h1] at *
42     <.> linarith
43     exact h_main
44
45 /--Consider three positive real numbers  $| (x |)$ ,  $| (y |)$ , and  $| (z |)$  such that  $| (x > 0$ 
46  $\rightarrow |)$ ,  $| (y > 0 |)$ , and  $| (z > 0 |)$ . Prove that the product of th
47 e expressions  $| (x - y + z) |$ ,  $| (y - z + x) |$ , and  $| (z - x + y) |$  is less than or
48  $\rightarrow$  equal to the product  $| (x \cdot y \cdot z) |$ .-/
49 theorem schur_like_ineq (x y z : ℝ) (hx : 0 < x) (hy : 0 < y) (hz : 0 < z) :
50    $(x - y + z) * (y - z + x) * (z - x + y) \leq x * y * z := by$ 
51   have h_main :  $(x - y + z) * (y - z + x) * (z - x + y) \leq x * y * z := by$ 
52     nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
53       mul_nonneg hx.le hy.le, mul_nonneg hy.le hz.le, mul_nonneg hz.le hx.le,
54       mul_nonneg (sq_nonneg (x - y)) (sq_nonneg (y - z)),
55       mul_nonneg (sq_nonneg (y - z)) (sq_nonneg (z - x)),
56       mul_nonneg (sq_nonneg (z - x)) (sq_nonneg (x - y)),
57       mul_nonneg (sq_nonneg (x - y + z)) (sq_nonneg (y - z + x)),
58       mul_nonneg (sq_nonneg (y - z + x)) (sq_nonneg (z - x + y)),
59       mul_nonneg (sq_nonneg (z - x + y)) (sq_nonneg (x - y + z)),
60       mul_nonneg (sq_nonneg (x + y - z)) (sq_nonneg (y + z - x)),
61       mul_nonneg (sq_nonneg (y + z - x)) (sq_nonneg (z + x - y)),
62       mul_nonneg (sq_nonneg (z + x - y)) (sq_nonneg (x + y - z))]
63   exact h_main
64
65 /--Consider positive real numbers  $| (a, b, c, x, y, z |)$  such that  $| (a \cdot b \cdot c =$ 
66  $\rightarrow 1 |)$  and  $| (a = \frac{x}{y} |)$ ,  $| (b = \frac{y}{z} |)$ ,  $| (c =$ 
67  $\frac{z}{x} |)$ . Prove that the inequality  $| ((a - 1 + \frac{1}{b}) \cdot (b - 1 +$ 
68  $\frac{1}{c}) \cdot (c - 1 + \frac{1}{a}) \leq 1 |)$  is equivalent to th
69 e inequality  $| ((x - y + z) \cdot (y - z + x) \cdot (z - x + y) \leq x \cdot y \cdot z) |$ .-/
70 theorem inequality_equivalence_under_parametrization (a b c x y z : ℝ)
71   (ha : 0 < a) (hb : 0 < b) (hc : 0 < c) (habc : a * b * c = 1)
72   (hx : 0 < x) (hy : 0 < y) (hz : 0 < z)
73   (hax : a = x / y) (hby : b = y / z) (hcz : c = z / x) :
74    $(a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) \leq 1 \leftrightarrow$ 
75    $(x - y + z) * (y - z + x) * (z - x + y) \leq x * y * z := by$ 
76   have h_main :  $(a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) = ((x + z - y) / y) *$ 
77      $\rightarrow ((x + y - z) / z) * ((y + z - x) / x) := by$ 

```

```

73   have h1 : a - 1 + 1 / b = (x + z - y) / y := by
74     have h1 : a = x / y := by linarith
75     have h2 : b = y / z := by linarith
76     rw [h1, h2]
77     field_simp [ha.ne', hb.ne', hx.ne', hy.ne', hz.ne']
78     <=> ring_nf
79     <=> field_simp [ha.ne', hb.ne', hx.ne', hy.ne', hz.ne']
80     <=> nlinarith
81   have h2 : b - 1 + 1 / c = (x + y - z) / z := by
82     have h1 : b = y / z := by linarith
83     have h2 : c = z / x := by linarith
84     rw [h1, h2]
85     field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
86     <=> ring_nf
87     <=> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
88     <=> nlinarith
89   have h3 : c - 1 + 1 / a = (y + z - x) / x := by
90     have h1 : c = z / x := by linarith
91     have h2 : a = x / y := by linarith
92     rw [h1, h2]
93     field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
94     <=> ring_nf
95     <=> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
96     <=> nlinarith
97   rw [h1, h2, h3]
98   <=> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
99   <=> ring_nf
100  <=> field_simp [ha.ne', hb.ne', hc.ne', hx.ne', hy.ne', hz.ne']
101  <=> nlinarith
102
103  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
104    have h1 : 0 < x * y := by positivity
105    have h2 : 0 < y * z := by positivity
106    have h3 : 0 < z * x := by positivity
107    have h4 : 0 < x * y * z := by positivity
108    constructor
109    · intro h
110      have h5 : ((x + z - y) / y) * ((x + y - z) / z) * ((y + z - x) / x) ≤ 1 := by
↪ linarith
111      have h6 : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by
112        field_simp at h5
113        rw [div_le_one (by positivity)] at h5
114        nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
115          mul_nonneg hx.le hy.le, mul_nonneg hy.le hz.le, mul_nonneg hz.le hx.le,
116          mul_nonneg (sq_nonneg (x - y)) hz.le, mul_nonneg (sq_nonneg (y - z)) hx.le,
117          mul_nonneg (sq_nonneg (z - x)) hy.le]
118      linarith
119    · intro h
120      have h5 : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z := by linarith
121      have h6 : ((x + z - y) / y) * ((x + y - z) / z) * ((y + z - x) / x) ≤ 1 := by
122        field_simp
123        rw [div_le_one (by positivity)]
124        nlinarith [sq_nonneg (x - y), sq_nonneg (y - z), sq_nonneg (z - x),
125          mul_nonneg hx.le hy.le, mul_nonneg hy.le hz.le, mul_nonneg hz.le hx.le,
126          mul_nonneg (sq_nonneg (x - y)) hz.le, mul_nonneg (sq_nonneg (y - z)) hx.le,
127          mul_nonneg (sq_nonneg (z - x)) hy.le]
128      linarith
129

```

```

130 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
131   rw [h_main]
132   rw [h_equiv]
133   <.>
134   simp_all
135   <.>
136   field_simp
137   <.>
138   ring_nf
139   <.>
140   nlinarith
141
142 exact h_final
143
144 theorem imo2000_p2
145   (a b c : ℝ) (ha : 0 < a) (hb : 0 < b) (hc : 0 < c)
146   (habc : a * b * c = 1) :
147   (a - 1 + 1 / b) * (b - 1 + 1 / c) * (c - 1 + 1 / a) ≤ 1 := by
148   -- 1. Parametrize x y z using positive numbers
149   obtain ⟨x, y, z, hx, hy, hz, ha_eq, hb_eq, hc_eq⟩ :=
150     imo2000_p2_existence_of_xyz a b c ha hb hc habc
151   -- 2. Use an equivalent lemma to transform the goal into the form involving x y z
152   have h_equiv :=
153     (inequality_equivalence_under_parametrization
154       (a := a) (b := b) (c := c) (x := x) (y := y) (z := z)
155       ha hb hc habc hx hy hz ha_eq hb_eq hc_eq)
156   -- 3. The Schur-type inequality yields the conclusion on the right-hand side.
157   have hxyz : (x - y + z) * (y - z + x) * (z - x + y) ≤ x * y * z :=
158     schur_like_ineq x y z hx hy hz
159   -- 4. Derive the original conclusion by reversing the equivalent proposition.
160   exact h_equiv.mpr hxyz

```