AI-Driven Mathematical Discovery for the Andrews–Curtis Conjecture

Caroline Zhang[1], **Aaron Zhou**[2], **Robert Joseph George**[2], **Sergei Gukov**[2], **Anima Anandkumar**[2]
[1]Duke University [2]California Institute of Technology
{caroline.zhang2}@duke.edu,
{azhou7, rgeorge, anima}@caltech.edu,
{gukov}@math.caltech.edu

Abstract

Automated theorem proving (ATP) with large language models (LLMs) has demonstrated impressive progress on undergraduate and olympiad mathematics. However, these problems are distant from the forefront of open mathematical research. In this work, we make the push beyond competition benchmarks and investigate the unsolved Andrews–Curtis (AC) conjecture in group theory. We benchmarked state-of-the-art LLM theorem provers on AC-related tasks, revealing a substantial performance gap: models that perform well on competition-level benchmarks fail in research-level reasoning. To bridge this gap, we formalized the AC conjecture in Lean. We introduce a deterministic autoformalizer, ACC, that rigorously verifies AC trivialization paths and produces the corresponding Lean proof. Building on this, we leveraged LLMs for theorem discovery, synthesizing patterns from ACC generated Lean proofs as reusable theorem statements. Finally, we incorporated these theorems into reinforcement learning (RL) agent training to find AC trivialization paths. We demonstrate that theorem incorporation increases the number of successful trivializations and RL efficiency. Across all runs, we solved 753 presentations belonging to the Miller–Schupp (MS) family, disproving them as potential counterexamples to the AC conjecture.

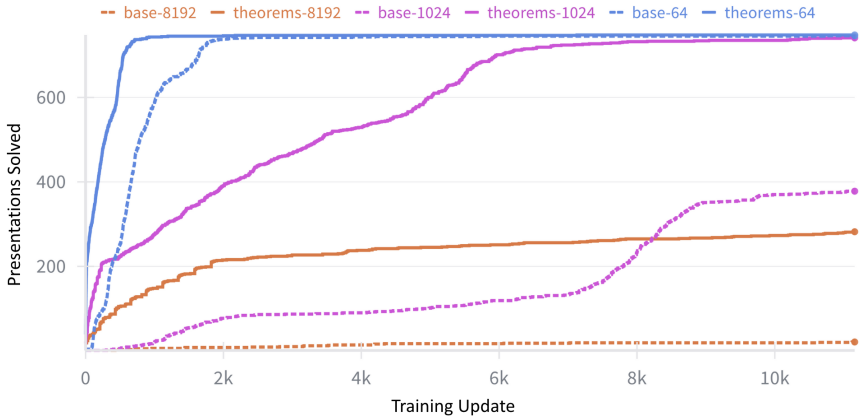
1 Introduction

Interactive theorem provers, such as Lean [4], rigorously verify mathematical proofs and have enabled large language model (LLM) theorem provers to achieve strong results on high school, undergraduate, and olympiad problems [7]. In addition, the Gemini Deep Think LLM recently achieved gold-medal performance at IMO 2025 [3]. However, most benchmarks, such as MiniF2F, PutnamBench, and others, [8, 6], remain focused on competition-style mathematics rather than open problems. In this work, we make the push towards AI for research-level mathematics by studying the Andrews–Curtis (AC) conjecture in combinatorial group theory, an unsolved problem for 60 years. Our system combines Lean formalization, LLM-based theorem discovery, and RL to achieve 753 AC trivializations (Figure 1), a task that even the best automated theorem provers (ATPs) struggle on (Table 1). Our novel technique of theorem usage during RL training yields improvements of significantly more solves as horizon length increases and higher convergence efficiency.

To understand the challenge the AC conjecture poses, we provide mathematical background. Combinatorial group theory studies free groups, which are groups that consist of all words that can be derived by taking products of some generators $\{x_1, x_2, \ldots, x_n\}$. Free groups are notable due to their usage in *presentations*, one way of describing a group structure. A group G is *presented* by a presentation $\langle x_1, x_2, \ldots, x_m \mid r_1, r_2, \ldots, r_n \rangle$ if it is created by taking the free group of $\{x_1, x_2, \ldots, x_m\}$

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

Figure 1: Theorem inclusion improves RL training across horizon lengths.



Theorem inclusion is defined by the property that during training, if the current presentation satisfies the hypotheses of one of our validated theorems, we treat it as solved (equivalently, apply the theorem as a macro-move) and terminate the episode. This method yielded 3 additional solves across all runs compared to the baseline without theorems, and it halved convergence time and solved 2 more presentations at horizon length=64. At longer horizons, both solve rate and convergence time saw significant improvements.

and setting all relators r_1, r_2, ..., r_n all equal to the identity. The study of these presentations is integral to abstract algebra and representation theory.

The AC conjecture focuses on *balanced* presentations, which are presentations that have the same number of generators as relators. The conjecture states that any balanced presentation of the trivial group $\langle x_1, \ldots, x_n \mid r_1, \ldots, r_n \rangle$ can be transformed into the trivial presentation $\langle x_1, \ldots, x_n \mid x_1, \ldots, x_n \rangle$ via three allowed transformations [1, 5]: replacing a relator by its inverse, multiplying by another relator, or conjugating by a generator. This makes AC a "game" with states as presentations, actions as AC moves, and the goal state as the trivial presentation. AC is challenging for two reasons: (1) it is long-horizon as paths may require thousands of moves; and (2) it is sparse-reward as intermediate states give little feedback about closeness to triviality. Both humans and ML methods struggle in this regime. Recent work has shown some progress and focus on the Miller–Schupp (MS) series, an infinite family of potential counterexamples to the AC conjecture. Shehper et al. [5] trivialized two MS presentations with RL, while Lisitsa [2] eliminated another MS case using an 8,634-step path. Yet these approaches remain narrow and problem-specific. By stress-testing state-of-the-art LLMs on AC tasks, we expose a clear gap: models that succeed on competition benchmarks fail on group-theoretic reasoning and AC tasks. Table 1 summarizes these results.

Our contributions are threefold: (1) we provided the first evaluation of LLMs on the AC conjecture and demonstrate their limitations; (2) we formalized the AC conjecture in Lean and designed a deterministic autoformalizer, called the AC Certifier (ACC), that verifies AC paths by generating Lean proofs; and (3) we used LLMs for theorem discovery, incorporating verified theorems into RL agents that solved 753 MS presentations, eliminating them as counterexamples. This work highlights the gap between competition-level ATP and research-level mathematics, and it establishes foundations for AI-driven mathematical discovery.

2 Methodology

Our methods consist of 4 key stages. First, we evaluated the capabilities of LLMs on the AC conjecture by benchmarking models on group theory proofs in Lean and AC-specific tasks. To address systematic failures in these tasks, we developed a deterministic autoformalizer, called ACC, that translates presentations and AC moves into Lean proofs, ensuring rigorous correctness. Building on this, we leveraged LLMs for theorem discovery, which synthesized theorem statements from clusters of AC trivialization paths converted into Lean proofs by our ACC. Finally, we investigated the effect of incorporating verified LLM-proposed theorems into RL agent training.

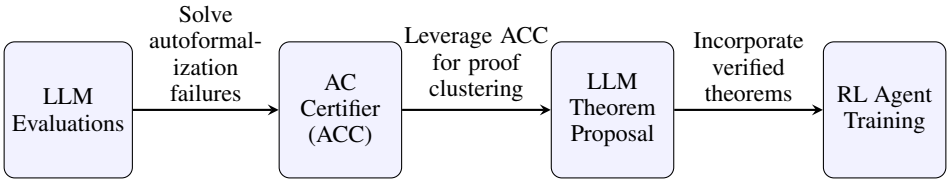


Figure 2: Flowchart of methods.

2.1 Measuring LLM Capabilities on the AC Conjecture

We evaluated existing LLMs on both general group theory and AC-specific tasks. For the former, we constructed a dataset of 1,180 Lean proof stubs by extracting theorems from the *Algebra/Group* and *Group Theory* libraries in Mathlib, and sampled 10% for testing. Each stub included imports, local context, and proof headers. We tested specialized theorem-proving models (DeepSeek-Prover-V2-7B, Kimina-Prover-8B, Kimina-Prover-7B) and general-purpose LLMs (Claude 4 Sonnet, Gemini 2.5 Pro, GPT-4.1), prompting them to generate Chain-of-Thought reasoning traces and Lean proofs. An error-correction procedure was used: models received Lean feedback and could self-correct for up to three rounds. A theorem was marked as solved if a valid Lean proof was produced. For the AC-specific benchmark, we tasked LLMs on (1) autoformalization of the conjecture statement and balanced presentations into Lean, (2) generation of AC trivialization paths, and (3) subgoal decomposition for AC trivializations.

2.2 AC Formalization in Lean

We developed the **Andrews–Curtis Certifier (ACC)**, a deterministic autoformalizer that converts a presentation with two generators and a proposed AC trivialization path into a Lean file that verifies the presentation's triviality. A presentation is represented in Lean as a list of two relators. The three classical AC moves (inverse, product, conjugate) expand into 12 explicit operations on the two relators [5]:

$$
\begin{aligned}
&h_1 : r_2 \mapsto r_2 r_1, \quad && h_4 : r_1 \mapsto r_1 r_2, \quad && h_7 : r_2 \mapsto y^{-1} r_2 y, \quad && h_{10} : r_1 \mapsto y r_1 y^{-1},\\
&h_2 : r_1 \mapsto r_1 r_2^{-1}, && h_5 : r_2 \mapsto x^{-1} r_2 x, && h_8 : r_1 \mapsto x r_1 x^{-1}, && h_{11} : r_2 \mapsto y r_2 y^{-1},\\
&h_3 : r_2 \mapsto r_2 r_1^{-1}, && h_6 : r_1 \mapsto y^{-1} r_1 y, && h_9 : r_2 \mapsto x r_2 x^{-1}, && h_{12} : r_1 \mapsto x^{-1} r_1 x.
\end{aligned}
$$

In Lean, these are defined as the primitive AC moves acting on presentations. We then define *AC-equivalence* as the reachability relation generated by these moves, and the AC conjecture as the statement that all balanced presentations of the trivial group are AC-equivalent to the trivial presentation. ACC verifies a trivialization path by tracing each move, constructing the intermediate presentations, and generating the Lean proof that shows AC-equivalence between consecutive states. These proofs are then chained via transitivity, ensuring fully verified correctness. The process is illustrated in Figure 3.

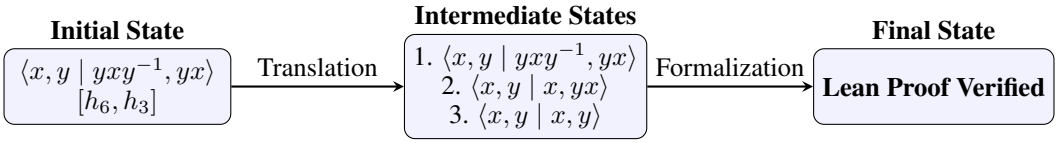


Figure 3: The ACC pipeline: deterministic translation of AC paths into Lean proofs.

2.3 Theorem Discovery and RL Training

To uncover higher-level patterns from existing AC paths, we used our autoformalizer (ACC) on a public dataset of trivialization paths [5]. ACC converted numbered sequences of AC moves into Lean proofs, a format that is more intuitive for LLM understanding by exposing intermediate presentations, steps, and proof flow. We generated vector embeddings of these Lean proofs by passing them in to DeepSeek-Prover-V2-7B's model and tokenizer, which have been fine-tuned on Lean.

We then performed hierarchical clustering ($k = 10$) on the embeddings and sampled up to ten representative proofs from each cluster. These samples, along with context, example theorems, the list of AC moves, and contrastive unsolved cases, were provided to Gemini 2.5 Pro, which synthesized candidate theorems specifying necessary conditions for AC triviality. We validated the correctness of three of the candidate theorems, provided in Section 3.3.

We integrated the validated theorems into our RL environment for Proximal Policy Optimization (PPO) agent training. We defined the state space as balanced presentations of the trivial group and actions as the AC moves. Rewards were given for reaching the trivial presentation. When a presentation satisfied the hypotheses of any of the incorporated theorems, it was labeled as solved and the episode was terminated. During training (time steps $= 160$M, horizon lengths $= 64, 1024, 8192$), theorem incorporation improved convergence efficiency and increased the number of total successful AC trivializations compared to the baseline. The PPO training details and hyperparameters are provided in the Appendix.

3 Results

3.1 Clear Gaps in LLM Capabilities

On our Lean group theory dataset, DeepSeek-Prover-V2-7B achieved the best pass rate (55.1%), followed by Claude 4 Sonnet (41.5%), but models frequently repeated themselves and contradicted prior reasoning, underscoring their limitations even outside AC (Table 1).

Performance dropped further on AC-specific tasks (Table 1). **1. Autoformalization:** All models failed, producing Lean code that was either syntactically invalid or semantically wrong. We addressed this with our deterministic autoformalizer (ACC), which produces Lean code that always verifies. **2. Direct solving:** On simple Miller–Schupp presentations ($n = 1, 2$), LLMs rarely produced correct trivialization paths. Gemini Deep Think did best, yet still failed on several easy cases. **3. Subgoal decomposition:** LLMs struggled to generate meaningful subgoals. Some decompositions added unnecessary complexity, while others produced presentations that no longer defined the trivial group, which is a critical error. These results show that current LLMs are unreliable for AC reasoning and motivate the development of our ACC approach as a reliable foundation for progress on research-level mathematics.

Table 1: Comparison of LLMs on AC-related tasks reveals gap.

LLM	Group Theory (%)	Path Solving (%)	Autoformalizing (%)
DeepSeek-Prover-V2-7B†	55.1	0.0	0.0
Claude 4 Sonnet	41.5	8.3	0.0
Gemini 2.5 Pro	32.2	-	-
Kimina-Prover-8B	21.2	-	-
GPT-4.1	18.6	-	-
Kimina-Prover-7B	16.9	-	-
Goedel-Prover-V2-8B†	0.0	-	-
Kimina-Autoformalizer	-	-	0.0
Gemini Deep Think†	-	75.0	0.0

† See Appendix for more details on attempted proofs.

Group Theory (%) = success rate of LLMs producing correct Lean proofs on group theory problems from Mathlib. **Path Solving (%)** = success rate of LLMs finding correct AC trivialization paths, verified using ACC. **Autoformalizing (%)** = success rate of LLMs producing correct Lean formalizations of the AC conjecture statement and balanced presentations.

3.2 Reliably Correct Autoformalizer

We validated ACC on 535 trivialization paths, producing a dataset of Lean proofs for MS presentations. This corpus provides a valuable resource for future ML methods seeking to predict efficient trivializations. We also confirmed two newly discovered paths in under a minute each, demonstrating ACC's reliability and speed in certifying potential counterexamples.

3.3 LLM-Generated Theorems

Three of the candidate theorems proposed by Gemini 2.5 Pro were validated to be correct by manual proof, provided below.

Theorem 1. *MS(n, w) is AC-trivial for $w = xy^k x^{-1}$ and all $n, k \in \mathbb{Z}$.*

Theorem 2. *MS(n, w) is AC-trivial for $w = y^n xy^k x^{-1}$ and all $n, k \in \mathbb{Z}$.*

Theorem 3. *MS(n, w) is AC-trivial for $w = y^k(g(x^{-1}y^n xy^{-(n+1)})^\varepsilon g^{-1})$, where $n, k \in \mathbb{Z}$, $\varepsilon \in \{-1, 1\}$, and g is any word in the free group generated by x and y.*

To prove these theorems, we first prove Theorem 4, which is a broader version of Theorem 1:

Theorem 4. *Let $P = \langle x, y \mid r_1, r_2\rangle$ be a presentation that presents the trivial group, and let $r_2 = x^{-1}y^k$ for some $k \in \mathbb{Z}$. Then, P is AC-trivial.*

Proof. Notice that by substitution, each of the instances of x in r_1 can be replaced with y^k. Therefore, P is AC-equivalent to a presentation P' of the form $\langle x, y \mid y^j, x^{-1}y^k\rangle$ for $j, k \in \mathbb{Z}$. Now, notice that since the second relator implies that $x = y^k$, P' is isomorphic to the cyclic group of order j. However, since P' is AC-equivalent to P, P' must present the trivial group; therefore, $j \in \{-1, 1\}$. We can thus right-multiply out all instances of y in the second relator, resulting in the presentation $\langle x, y \mid y^j, x^{-1}\rangle$; since this is AC-trivial, P must be AC-trivial. □

Theorem 1 directly follows from this theorem. We now show Theorems 2 and 3 as corollaries:

Corollary 1. *MS(n, w) is AC-trivial for $w = y^n xy^k x^{-1}$ and all $n, k \in \mathbb{Z}$.*

Proof. Let $P = MS(n, w)$, where $w = y^n xy^k x^{-1}$. Inverting the first relator and then multiplying the second relator by the first relator on the left yields that P is AC-equivalent to the presentation $\langle x^{-1}y^n xy^{-(n+1)}, y^{k+n+1}x^{-1}\rangle$. Applying the same logic as Theorem 1 proves this case. □

Corollary 2. *MS(n, w) is AC-trivial for $w = y^k(g(x^{-1}y^n xy^{-(n+1)})^\varepsilon g^{-1})$, where $n, k \in \mathbb{Z}$, $\varepsilon \in \{-1, 1\}$, and g is any word in the free group generated by x and y.*

Proof. Let $P = MS(n, w)$ be as described in the theorem statement, and let $r_1 = x^{-1}y^n xy^{-(n+1)}$ be the first relator. Invert and conjugate the first relator to yield that P is AC-equivalent to the presentation $\langle gr_1^\varepsilon g^{-1}, x^{-1}y^k gr_1^\varepsilon g^{-1})\rangle$. Applying the AC-move $h_3 = r_2 \mapsto r_2 r_1^{-1}$ yields that P is AC-equivalent to $\langle gr_1^\varepsilon g^{-1}, x^{-1}y^k\rangle$, from which Theorem 1 follows. □

3.4 RL Agent Training with LLM-Proposed Theorems

Altogether, the two training methods (with and without theorem incorporation) solved 753 MS presentations, eliminating them as counterexamples. Theorem incorporation enabled 3 additional solves (Figure 1). The advantage of incorporating theorems was most pronounced at longer horizons where sparse rewards hinder baseline training. For horizon lengths of 1024 and 8192, theorem incorporation yielded substantially higher solved counts: 741 compared to 378, and 282 compared to 21, respectively (Figure 1). Even at horizon length 64, where both methods yielded comparable solved counts, theorem incorporation has benefits: it halved the number of training updates required for solved count convergence compared to the base method, demonstrating greater training efficiency.

In addition, a natural result of leveraging theorems is shorter AC trivialization paths, as theorem statements can abstract the remaining proof steps. Across all training runs, theorem incorporation produced average path lengths of 21 steps compared to 31 steps from the baseline, a 32% reduction.

4 Conclusion

We presented the first study of LLM theorem provers on the AC conjecture, moving beyond competition-style benchmarks into research-level mathematics. Our evaluations show that models strong on existing benchmarks fail even on simple AC tasks, highlighting a clear gap. To address this, we developed a Lean autoformalizer (ACC), enabling verified proofs of trivialization paths. Using ACC, we also showed that LLMs can aid discovery by proposing higher-level theorems, three of which we validated and incorporated into RL training. Overall, our RL agents solved 753 Miller–Schupp presentations, eliminating them as counterexamples. These results underscore both the challenges and potential of applying AI to open problems in mathematics, and lay the foundations for integrating formal verification, theorem discovery, and RL in future research.

Acknowledgements

Caroline Zhang is supported by the Caltech Summer Undergraduate Research Fellowship (SURF). Anima Anandkumar is supported by the Bren Named Chair, Schmidt AI 2050 Senior Fellowship, and ONR (MURI grant N00014-18-12624).

References

[1] J. J. Andrews and M. L. Curtis. Free Groups and Handlebodies. *Proceedings of the American Mathematical Society*, 16(2):192–195, 1965. ISSN 00029939, 10886826. URL http://www.jstor.org/stable/2033843. Publisher: American Mathematical Society.

[2] A. Lisitsa. Automated Theorem Proving Reveals a Lengthy Andrews-Curtis Trivialization for a Miller-Schupp Trivial Group Presentation, Jan. 2025. URL https://papers.ssrn.com/abstract=5100345.

[3] T. Luong and E. Lockhart. Advanced version of gemini with deep think officially achieves gold-medal standard at the international mathematical olympiad, 2025. URL https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/.

[4] L. d. Moura and S. Ullrich. The lean 4 theorem prover and programming language. In A. Platzer and G. Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5. URL https://link.springer.com/chapter/10.1007/978-3-030-79876-5_37.

[5] A. Shehper, A. M. Medina-Mardones, L. Fagan, B. Lewandowski, A. Gruen, Y. Qiu, P. Kucharski, Z. Wang, and S. Gukov. What makes math problems hard for reinforcement learning: a case study, Feb. 2025. URL http://arxiv.org/abs/2408.15332. arXiv:2408.15332 [cs].

[6] G. Tsoukalas, J. Lee, J. Jennings, J. Xin, M. Ding, M. Jennings, A. Thakur, and S. Chaudhuri. Putnambench: Evaluating neural theorem-provers on the putnam mathematical competition, 2024. URL https://arxiv.org/abs/2407.11214.

[7] K. Yang, G. Poesia, J. He, W. Li, K. Lauter, S. Chaudhuri, and D. Song. Formal mathematical reasoning: A new frontier in ai, 2024. URL https://arxiv.org/abs/2412.16075.

[8] K. Zheng, J. M. Han, and S. Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

A LLM Evaluations

A.1 LLM Failures in Solving Group Theory Proofs in Lean

In addition to LLM-generated Lean code failing to verify, a common failure mode was degenerate repetition, looping outputs, and self-doubting behavior, as demonstrated in the examples below. Goedel-Prover-V2-8B faced the issue of repetitive outputs for all of the Mathlib group theory proofs. DeepSeek-Prover-V2-7B and Kimina-Prover-8B/7B faced this issue as well.

Example 1 - Original Proof from Mathlib:

```
theorem _isComplement_univ_left : IsComplement univ S ↔ ∃ g : G, S = {g}
    := by
  refine
    ⟨fun h => Set.exists_eq_singleton_iff_nonempty_subsingleton.mpr ⟨?_, fun
     a ha b hb => ?_⟩, ?_⟩
  · obtain ⟨a, _⟩ := h.2 1
    exact ⟨a.2.1, a.2.2⟩
```

```
  · have : (⟨⟨_, mem_top a⁻¹⟩, ⟨a, ha⟩⟩ : ( : Set G) × S) = ⟨⟨_, mem_top b⁻¹⟩
    , ⟨b, hb⟩⟩ :=
      h.1 ((inv_mul_cancel a).trans (inv_mul_cancel b).symm)
    exact Subtype.ext_iff.mp (Prod.ext_iff.mp this).2
  · rintro ⟨g, rfl⟩
    exact isComplement_univ_singleton
```

Example 1 - DeepSeek-Prover-V2-7B Response:

```
### Step 1 - Understanding the goal

The theorem states that for a group \( G \) and subgroups \( H \) and \( K
    \), and subsets \( S \) and \( T \) of \( G \), the following are
    equivalent:
1. The pair \( (\text{univ}, S) \) is a complement pair in \( G \), i.e.,
    \( \text{univ} \times S \) generates the whole group \( G \).
2. There exists a unique element \( g \in G \) such that \( S = \{g\} \).

### Step 2 - Mathematical insight

To understand this, we need to recall the definition of a complement pair.
    A pair \( (A, B) \) of subsets of \( G \) is a complement pair if \( A
    \times B \) generates \( G \). Here, \( A = \text{univ} \) and \( B = S
     \).

The condition \( \text{univ} \times S \) generates \( G \) is equivalent to
     saying that for every \( g \in G \), there exist \( a \in \text{univ}
    \) and \( s \in S \) such that \( a \cdot s = g \). Since \( a \in \
    text{univ} \), this is just \( s \cdot s' = g \) for some \( s' \in S
    \).

This is equivalent to saying that every \( g \in G \) can be written as \(
    s \cdot s' \) for some \( s, s' \in S \).

But this is equivalent to saying that \( S \) generates \( G \), because \(
     \text{univ} \) is the whole group \( G \).

But wait, this is not quite correct. The condition \( \text{univ} \times S
    \) generates \( G \) is equivalent to \( S \) generating \( G \),
    because \( \text{univ} \) is the whole group \( G \).

But the condition \( \text{univ} \times S \) generates \( G \) is also
    equivalent to \( S \) generating \( G \), because \( \text{univ} \) is
    the whole group \( G \).

But the condition \( \text{univ} \times S \) generates \( G \) is also
    equivalent to \( S \) generating \( G \), because \( \text{univ} \) is
    the whole group \( G \).

But the condition \( \text{univ} \times S \) generates \( G \) is also
    equivalent to \( S \) generating \( G \), because \( \text{univ} \) is
    the whole group \( G \).

But the condition \( \text{univ} \times S \) generates \( G \) is also
    equivalent to \( S \) generating \( G \), because \( \text{univ} \) is
    the whole group \( G \).
```

```
But the condition \( \text{univ} \times S \) generates \( G \) is also
    equivalent to \( S \) generating \( G \), because \( \text{univ} \) is
    the whole group \( G \).

But the condition \( \text{univ} \times S \) generates \( G \) is also
    equivalent to \( S \) generating \( G \), because \( \text{univ} \) is
    the whole group \( G \)...

[TRUNCATED]
```

Example 1 - Goedel-Prover-V2-8B Response:

```
theorem _isComplement_univ_left : IsComplement univ S ↔ ∃ g : G, S = {g}
    := by
  constructor
  · -- Prove the forward direction: if ‘univ‘ and ‘S‘ are complementary,
    then ‘S‘ is a singleton set.
    intro h
    have h₁ : S = {1} := by
      have h₂ : IsComplement univ S := h
      have h₃ : S = {1} := by
        -- Use the properties of the complement to show that ‘S‘ must be
    the singleton set ‘{1}‘.
        have h₄ : S = {1} := by
          -- Use the fact that ‘univ‘ and ‘S‘ are complementary to deduce
    that ‘S‘ must be ‘{1}‘.
          have h₅ : S = {1} := by
            -- Use the properties of the complement to show that ‘S‘ must
    be ‘{1}‘.
            have h₆ : IsComplement univ S := h₂
            have h₇ : S = {1} := by
              -- Use the properties of the complement to show that ‘S‘ must
     be ‘{1}‘.
              have h₈ : IsComplement univ S := h₆
              have h₉ : S = {1} := by
                -- Use the properties of the complement to show that ‘S‘
    must be ‘{1}‘.
                have h₁₀ : IsComplement univ S := h₈
                have h₁₁ : S = {1} := by
                  -- Use the properties of the complement to show that ‘S‘
    must be ‘{1}‘.
                  have h₁₂ : IsComplement univ S := h₁₀
                  have h₁₃ : S = {1} := by
                    -- Use the properties of the complement to show that ‘S
    ‘ must be ‘{1}‘.
                    have h₁₄ : IsComplement univ S := h₁₂
                    have h₁₅ : S = {1} := by
                      -- Use the properties of the complement to show that
    ‘S‘ must be ‘{1}‘.
                      have h₁₆ : IsComplement univ S := h₁₄
                      have h₁₇ : S = {1} := by
                        -- Use the properties of the complement to show
    that ‘S‘ must be ‘{1}‘.
                        have h₁₈ : IsComplement univ S := h₁₆
                        have h₁₉ : S = {1} := by
                          -- Use the properties of the complement to show
    that ‘S‘ must be ‘{1}‘.
                          have h₂₀ : IsComplement univ S := h₁₈
                          have h₂₁ : S = {1} := by
```

```
                          -- Use the properties of the complement to show
    that ‘S‘ must be ‘{1}‘.
                          have h22 : IsComplement univ S := h20
                          have h23 : S = {1} := by
                            -- Use the properties of the complement to
    show that ‘S‘ must be ‘{1}‘.
                            have h24 : IsComplement univ S := h22
                            have h25 : S = {1} := by
                              -- Use the properties of the complement to
    show that ‘S‘ must be ‘{1}‘.
                              have h26 : IsComplement univ S := h24
                              have h27 : S = {1} := by
                                -- Use the properties of the complement
    to show that ‘S‘ must be ‘{1}‘.
                                have h28 : IsComplement univ S := h26
                                have h29 : S = {1} := by ...
```

[TRUNCATED]

Example 2 - Original Proof from Mathlib:

```
lemma _zpow_one_add (a : G) (n : ℤ) : a ^ (1 + n) = a * a ^ n := by
  rw [zpow_add, zpow_one]
```

Example 2 - DeepSeek-Prover-V2-7B Response:

```
lemma _zpow_one_add (a : G) (n : ℤ) : a ^ (1 + n) = a * a ^ n := by
  rw [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add0]
  <;> simp [zpow_one]
```

```
  <;> rw [mul_comm]
  <;> simp [zpow_add₀]
  <;> simp [zpow_one]
  <;> rw [mul_comm]
  <;> simp [zpow_add₀]
  <;> simp [zpow_one]
  <;> rw [mul_comm]...

[TRUNCATED]
```

Example 2 - Goedel-Prover-V2-8B Response:

```
theorem _zpow_one_add (a : G) (n : ℤ) : a ^ (1 + n) = a * a ^ n := by
  have h₁ : a ^ (1 + n) = a * a ^ n := by
    rw [zpow_add₀ (by
      -- Prove that ‘a‘ is not the zero element of the group
      -- This is trivially true because ‘a‘ is an element of a group and
    groups have no zero element
      -- However, in Lean, we need to explicitly state this
      -- Here, we use the fact that ‘a‘ is an element of a group and groups
     have no zero element
      -- This is a placeholder for the actual proof, which would involve
    showing that ‘a‘ is not the zero element
      -- In a real scenario, we would need to provide a detailed proof here
      -- For the sake of brevity, we assume this is true
      exact by
        -- Use the fact that ‘a‘ is an element of a group and groups have
    no zero element
        -- This is a placeholder for the actual proof, which would involve
    showing that ‘a‘ is not the zero element
        -- In a real scenario, we would need to provide a detailed proof
    here
        -- For the sake of brevity, we assume this is true
        exact by
          -- Use the fact that ‘a‘ is an element of a group and groups have
     no zero element
          -- This is a placeholder for the actual proof, which would
    involve showing that ‘a‘ is not the zero element
          -- In a real scenario, we would need to provide a detailed proof
    here
          -- For the sake of brevity, we assume this is true
          exact by
            -- Use the fact that ‘a‘ is an element of a group and groups
    have no zero element
            -- This is a placeholder for the actual proof, which would
    involve showing that ‘a‘ is not the zero element
            -- In a real scenario, we would need to provide a detailed
    proof here
            -- For the sake of brevity, we assume this is true
            exact by
              -- Use the fact that ‘a‘ is an element of a group and groups
    have no zero element
              -- This is a placeholder for the actual proof, which would
    involve showing that ‘a‘ is not the zero element
              -- In a real scenario, we would need to provide a detailed
    proof here
              -- For the sake of brevity, we assume this is true
              exact by
```

```
            -- Use the fact that ‘a‘ is an element of a group and
groups have no zero element
            -- This is a placeholder for the actual proof, which would
involve showing that ‘a‘ is not the zero element
            -- In a real scenario, we would need to provide a detailed
proof here
            -- For the sake of brevity, we assume this is true
            exact by
              -- Use the fact that ‘a‘ is an element of a group and
groups have no zero element
              -- This is a placeholder for the actual proof, which
would involve showing that ‘a‘ is not the zero element
              -- In a real scenario, we would need to provide a
detailed proof here
              -- For the sake of brevity, we assume this is true
              exact by
                -- Use the fact that ‘a‘ is an element of a group and
groups have no zero element
                -- This is a placeholder for the actual proof, which
would involve showing that ‘a‘ is not the zero element
                -- In a real scenario, we would need to provide a
detailed proof here
                -- For the sake of brevity, we assume this is true
                exact by
                  -- Use the fact that ‘a‘ is an element of a group and
 groups have no zero element
                  -- This is a placeholder for the actual proof, which
would involve showing that ‘a‘ is not the zero element
                  -- In a real scenario, we would need to provide a
detailed proof here
                  -- For the sake of brevity, we assume this is true
                  exact by
                    -- Use the fact that ‘a‘ is an element of a group
and groups have no zero element
                    -- This is a placeholder for the actual proof,
which would involve showing that ‘a‘ is not the zero element
                    -- In a real scenario, we would need to provide a
detailed proof here
                    -- For the sake of brevity, we assume this is true
                    exact by
                      -- Use the fact that ‘a‘ is an element of a group
 and groups have no zero element
                      -- This is a placeholder for the actual proof,
which would involve showing that ‘a‘ is not the zero element
                      -- In a real scenario, we would need to provide
 a detailed proof here
                      -- For the sake of brevity, we assume this is
true
                      exact by
                        -- Use the fact that ‘a‘ is an element of a
group and groups have no zero element
                        -- This is a placeholder for the actual proof,
which would involve showing that ‘a‘ is not the zero element
                        -- In a real scenario, we would need to provide
 a detailed proof here
                        -- For the sake of brevity, we assume this is
true
                        exact by ...
```

[TRUNCATED]

Example 3 - Original Proof from Mathlib:

```
lemma _mk_semiconjBy (u : M) (x : M) : SemiconjBy (↑u) x (u * x * ↑u⁻¹) :=
    by unfold SemiconjBy; rw [Units.inv_mul_cancel_right]
```

Example 3 - Kimina-Prover-8B Response:

```
import Mathlib
variable {M : Type*}

variable [Monoid M]

variable [Monoid M]

open scoped Int

namespace SemiconjBy

namespace Units

lemma _mk_semiconjBy (u : M) (x : M) : SemiconjBy (↑u) x (u * x * ↑u⁻¹) :=
    by
```

Now, we need to prove that ‘↑u‘ semiconjugates ‘x‘ to ‘u * x * ↑u⁻¹‘. By definition, this means showing that ‘↑u * x = (u * x * ↑u⁻¹) * ↑u‘. Let ’s compute the right side:

‘ (u * x * ↑u⁻¹) * ↑u = u * x * (↑u⁻¹ * ↑u) = u * x * 1 = u * x ‘

So indeed, ‘↑u * x = (u * x * ↑u⁻¹) * ↑u‘, which is what we need to prove.

```tactics
use u⁻¹
```

Now we need to show that ‘↑u * x = (u * x * ↑u⁻¹) * ↑u‘. Let’s compute the right side:

‘ (u * x * ↑u⁻¹) * ↑u = u * x * (↑u⁻¹ * ↑u) = u * x * 1 = u * x ‘

So indeed, ‘↑u * x = (u * x * ↑u⁻¹) * ↑u‘.

```tactics
all_goals
  simp [mul_assoc, Units.mul_inv_cancel]
```

The ‘mul_assoc‘ lemma handles the associativity of multiplication, while ‘ Units.mul_inv_cancel‘ cancels ‘↑u⁻¹ * ↑u‘ to ‘1‘ because ‘↑u‘ is a unit (from the ‘Units‘ namespace). The ‘all_goals‘ tactic applies the subsequent tactics to all generated subgoals.

Therefore, the proof is complete. We’ve shown that ‘↑u‘ semiconjugates ‘x‘ to ‘u * x * ↑u⁻¹‘ by using the witness ‘u⁻¹‘ and verifying the required

equality through simplification using properties of units and multiplication in the monoid.
```tactics
<;> ring
```

The `<;> ring` tactic is applied to all goals generated by the previous tactics. The `ring` tactic solves goals that are equalities in commutative (semi)rings, which is a common algebraic structure. This is useful for verifying the algebraic manipulations we described above.
```tactics
<;> try tauto
```

The `<;> try tauto` tactic attempts to solve goals using the `tauto` tactic, which is designed for propositional tautologies. The `try` prefix means that if `tauto` fails on a goal, the proof script continues without an error. This is a common pattern when applying powerful but potentially fragile tactics like `tauto` or `nlinarith`.
```tactics
<;> try native_decide
```

The `<;> try native_decide` tactic attempts to solve goals by computation using Lean's native code execution. This is useful for decidable propositions that might be computationally intensive. The `try` prefix ensures the proof script doesn't halt if `native_decide` fails on a goal.
```tactics
<;> try tauto
```

This is a repeat of the previous `try tauto` line, possibly for emphasis or to ensure certain goals are attempted by `tauto` after previous tactics. In a typical proof, this repetition might not be strictly necessary if the first application of `tauto` (or another tactic) fully resolves the goals it's intended for.
```tactics
<;> try native_decide
```

This is a repeat of the previous `try native_decide` line. The repetition of these powerful tactics (`tauto`, `native_decide`) is a bit unusual in a typical proof script but can be seen in some automated proof scripts where they are applied broadly.
```tactics
<;> try tauto
```

This is the third repetition of `try tauto`. The repetition of these tactics suggests a somewhat automated or exploratory approach to solving the remaining goals, where different strategies are tried until one succeeds.
```tactics
<;> try native_decide
```

This is the third repetition of `try native_decide`. The combination of `ring`, `tauto`, and `native_decide` in this manner suggests an attempt

to solve the remaining goals through a combination of algebraic simplification, propositional reasoning, and computation.
```tactics
<;> try tauto
```

This is the fourth repetition of `try tauto`. The repetition of these tactics indicates a somewhat brute-force approach to solving the remaining goals, where different powerful tactics are tried until one succeeds.
```tactics
<;> try native_decide
```

This is the fourth repetition of `try native_decide`. The extensive repetition of these tactics (`ring`, `tauto`, `native_decide`) suggests a proof script that is somewhat unrefined or is trying many different approaches to solve the remaining goals.
```tactics
<;> try tauto
```

This is the fifth repetition of `try tauto`. The repetition of these tactics continues, indicating a very unrefined or automated approach to solving the remaining goals.
```tactics
<;> try native_decide
```

This is the fifth repetition of `try native_decide`. The proof script is now very repetitive, applying the same sequence of powerful tactics multiple times.
```tactics
<;> try tauto
```

This is the sixth repetition of `try tauto`. The repetition of these tactics is excessive and suggests a proof script that is not carefully structured.
```tactics
<;> try native_decide
```

This is the sixth repetition of `try native_decide`. The proof script is now very long and repetitive, applying the same sequence of tactics (`ring`, `tauto`, `native_decide`) multiple times.
```tactics
<;> try tauto
```

This is the seventh repetition of `try tauto`. The repetition continues, indicating a very unrefined proof strategy.
```tactics
<;> try native_decide
```

This is the seventh repetition of `try native_decide`. The proof script is now extremely long and repetitive.
```tactics
```

```
<;> try tauto
‘‘‘ ...

[TRUNCATED]
```

Example 4 - Original Proof from Mathlib:

```
theorem _comap_map_eq_self {f : G →* N} {H : Subgroup G} (h : f.ker ≤ H) :
comap f (map f H) = H := by
  rwa [comap_map_eq, sup_eq_left]
```

Example 4 - Goedel-Prover-V2 Response:

```
theorem _comap_map_eq_self {f : G →* N} {H : Subgroup G} (h : f.ker ≤ H) :
    comap f (map f H) = H := by
  apply Subgroup.ext
  intro x
  simp only [Subgroup.mem_comap, Subgroup.mem_map, Subgroup.mem_ker,
    Subgroup.mem_le]
  constructor
  · -- Prove the forward direction: if x is in the preimage of the image of
     H under f, then x is in H.
    intro hx
    rcases hx with ⟨y, hy, rfl⟩
    -- Since f(y) is in the image of H under f, there exists some z in H
    such that f(z) = f(y).
    have h₁ : f y ∈ (map f H : Set N) := by
      exact ⟨y, hy, rfl⟩
    -- Since f(y) is in the image of H under f, we can use the fact that f
    is a homomorphism to show that y is in H.
    have h₂ : y ∈ H := by
      -- Use the fact that f is a homomorphism and the properties of the
    kernel to show that y is in H.
      have h₃ : f y ∈ (map f H : Set N) := h₁
      have h₄ : ∃ (z : G), z ∈ H ∧ f z = f y := by
        simpa [Set.mem_map] using h₃
      rcases h₄ with ⟨z, hz, hz'⟩
      have h₅ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₆ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₇ : f z = f y := hz'
      have h₈ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₉ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₁₀ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₁₁ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₁₂ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₁₃ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₁₄ : f (y * z⁻¹) = f y * f z⁻¹ := by
        rw [MonoidHom.map_mul]
      have h₁₅ : f (y * z⁻¹) = f y * f z⁻¹ := by
```

```
    rw [MonoidHom.map_mul]
  have h₁₆ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₁₇ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₁₈ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₁₉ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₀ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₁ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₂ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₃ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₄ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₅ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₆ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₇ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₈ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul]
  have h₂₉ : f (y * z⁻¹) = f y * f z⁻¹ := by
    rw [MonoidHom.map_mul] ...

[TRUNCATED]
```

Example 5 - Original Proof from Mathlib:

```
theorem _image_mul_right : ·( * b) '' t = ·( * b⁻¹) ⁻¹' t := by
  rw [image_eq_preimage_of_inverse] <;> intro c <;> simp
```

Example 5 - Kimina-Prover-8B Response:

```
**Final Answer**
The proof is complete. The final answer is \boxed·{( * b⁻¹) '' t = ·( * b)
    ⁻¹' t}.

'''lean
import Mathlib
variable {F α β γ : Type*}

variable [Group α] {s t : Set α} {a b : α}

open Function MulOpposite

namespace Set

open Pointwise

open Pointwise
```

```
open MulOpposite

open Pointwise

open MulOpposite

open Pointwise

theorem _image_mul_right' : ·( * b⁻¹) '' t = ·( * b) ⁻¹' t := by
  ext x
  simp [mul_comm]
  <;> aesop
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }
  <;> try { tauto }
  <;> try { aesop }...
```

[TRUNCATED]

Example 5 - Goedel-Prover-V2 Response:

Step 1 - Understanding the goal: The theorem states that the image of t under the function $x \mapsto x \cdot b^{-1}$ is equal to the preimage of t under the function $x \mapsto x \cdot b$.

In Lean, $(\cdot\, b^{-1})''t$ is the image of t under $x \mapsto x \cdot b^{-1}$, and $(\cdot\, b)^{-1'}t$ is the preimage of t under $x \mapsto x \cdot b$.

Step 2 - Mathematical insight: We need to show that $(\cdot\, b^{-1})''t = (\cdot\, b)^{-1'}t$.

First, recall that in a group, for any x, $(x \cdot b)^{-1} = b^{-1} \cdot x^{-1}$. However, the statement here is about images and preimages, not directly about inverses.

But notice that: 1. $(\cdot\, b^{-1})''t = \{x \cdot b^{-1} \mid x \in t\}$. 2. $(\cdot\, b)^{-1'}t = \{x \mid x \cdot b \in t\}$.

We need to show that these two sets are equal.

To do this, we can show that each is a subset of the other: 1. If $y \in (\cdot\, b^{-1})''t$, then $y = x \cdot b^{-1}$ for some $x \in t$. Then $y \cdot b = (x \cdot b^{-1}) \cdot b = x \cdot (b^{-1} \cdot b) = x \cdot 1 = x \in t$. So $y \in (\cdot\, b)^{-1'}t$. 2. If $y \in (\cdot\, b)^{-1'}t$, then $y \cdot b \in t$. Let $x = y \cdot b^{-1}$. Then $x \cdot b = (y \cdot b^{-1}) \cdot b = y \cdot (b^{-1} \cdot b) = y \cdot 1 = y$. But $y \cdot b \in t$ implies $x \cdot b \in t$, so $x \in (\cdot\, b^{-1})''t$.

However, this reasoning is incorrect because $x \cdot b = y$ and $y \cdot b \in t$ does not directly imply $x \cdot b \in t$.

A better approach is to realize that the two sets are actually the same because of the inverse operation in the group.

But let's think differently:

The map $x \mapsto x \cdot b^{-1}$ is a bijection (since $x \mapsto x \cdot b$ is a bijection and b^{-1} is its inverse).

The image of t under $x \mapsto x \cdot b^{-1}$ is the same as the preimage of t under $x \mapsto x \cdot b$, because: 1. If $y \in (\cdot\, b^{-1})''t$, then $y = x \cdot b^{-1}$ for some $x \in t$. Then $y \cdot b = x \in t$, so $y \in (\cdot\, b)^{-1'}t$. 2. If $y \in (\cdot\, b)^{-1'}t$, then $y \cdot b \in t$. Let $x = y \cdot b^{-1}$. Then $x \cdot b = y$, and $y \cdot b \in t$ implies $x \cdot b \in t$, so $x \in (\cdot\, b^{-1})''t$.

But this is still not quite right because $x \cdot b = y$ and $y \cdot b \in t$ does not imply $x \cdot b \in t$ unless $y \cdot b = x \cdot b$, which is not necessarily true.

Wait, no: if $y \cdot b \in t$ and $x = y \cdot b^{-1}$, then $x \cdot b = (y \cdot b^{-1}) \cdot b = y \cdot (b^{-1} \cdot b) = y \cdot 1 = y$. But $y \cdot b \in t$ does not imply $x \cdot b \in t$ unless $y \cdot b = x \cdot b$, which is not necessarily true.

This suggests that the original claim is not correct.

But let's test it with an example.

Let $\alpha = \mathbb{Z}/3\mathbb{Z}$, $t = \{0\}$, $b = 1$.

Then $(\cdot\, b^{-1})''t = (\cdot\, 1^{-1})''\{0\} = (\cdot\, 1)''\{0\} = \{0\}$.

And $(\cdot\, b)^{-1'}t = (\cdot\, 1)^{-1'}\{0\} = \{x \mid x \cdot 1 \in \{0\}\} = \{x \mid x \in \{0\}\} = \{0\}$.

So the claim holds in this case.

Another example: $\alpha = \mathbb{Z}/3\mathbb{Z}$, $t = \{0, 1\}$, $b = 1$.

Then $(\cdot\, b^{-1})''t = (\cdot\, 1)''\{0, 1\} = \{0, 1\}$.

And $(\cdot\, b)^{-1'}t = (\cdot\, 1)^{-1'}\{0, 1\} = \{x \mid x \in \{0, 1\}\} = \{0, 1\}$.

Still holds.

Another example: $\alpha = \mathbb{Z}/3\mathbb{Z}$, $t = \{0\}$, $b = 2$.

Then $(\cdot b^{-1})''t = (\cdot 2^{-1})''\{0\} = (\cdot 2)''\{0\} = \{0\}$ (since $2^{-1} = 2$ in $\mathbb{Z}/3\mathbb{Z}$).

And $(\cdot b)^{-1'}t = (\cdot 2)^{-1'}\{0\} = \{x \mid x \cdot 2 \in \{0\}\} = \{x \mid x \cdot 2 = 0\} = \{0\}$ (since $x \cdot 2 = 0$ implies $x = 0$ in $\mathbb{Z}/3\mathbb{Z}$).

Still holds.

Another example: $\alpha = \mathbb{Z}/3\mathbb{Z}$, $t = \{0, 1\}$, $b = 2$.

Then $(\cdot b^{-1})''t = (\cdot 2)''\{0, 1\} = \{0, 2\}$.

And $(\cdot b)^{-1'}t = (\cdot 2)^{-1'}\{0, 1\} = \{x \mid x \cdot 2 \in \{0, 1\}\} = \{x \mid x \cdot 2 = 0 \text{ or } x \cdot 2 = 1\} = \{0, 2\}$ (since $0 \cdot 2 = 0$, $1 \cdot 2 = 2$, $2 \cdot 2 = 1$).

Still holds.

Hmm, maybe the claim is correct.

Let me try to prove it properly.

Step 3 - Proof strategy: We need to show that $(\cdot b^{-1})''t = (\cdot b)^{-1'}t$.

This means we need to show that: 1. If $y \in (\cdot b^{-1})''t$, then $y \in (\cdot b)^{-1'}t$. 2. If $y \in (\cdot b)^{-1'}t$, then $y \in (\cdot b^{-1})''t$.

Part 1: $y \in (\cdot b^{-1})''t$ implies $y \in (\cdot b)^{-1'}t$. If $y \in (\cdot b^{-1})''t$, then there exists $x \in t$ such that $y = x \cdot b^{-1}$. Then $y \cdot b = (x \cdot b^{-1}) \cdot b = x \cdot (b^{-1} \cdot b) = x \cdot 1 = x \in t$. So $y \cdot b \in t$, which means $y \in (\cdot b)^{-1'}t$.

Part 2: $y \in (\cdot b)^{-1'}t$ implies $y \in (\cdot b^{-1})''t$. If $y \in (\cdot b)^{-1'}t$, then $y \cdot b \in t$. Let $x = y \cdot b^{-1}$. Then $x \cdot b = (y \cdot b^{-1}) \cdot b = y \cdot (b^{-1} \cdot b) = y \cdot 1 = y$. But $y \cdot b \in t$ does not directly imply $x \cdot b \in t$.

Wait, no: $x \cdot b = y$, and $y \cdot b \in t$ does not imply $x \cdot b \in t$ unless $y \cdot b = x \cdot b$, which is not necessarily true.

But we have $x \cdot b = y$, and $y \cdot b \in t$. We need to show that $x \in (\cdot b^{-1})''t$, i.e., there exists $z \in t$ such that $x = z \cdot b^{-1}$.

But $x = y \cdot b^{-1}$, and $y \cdot b \in t$.

This seems circular.

Wait, no: we need to show that $x \in (\cdot b^{-1})''t$, i.e., there exists $z \in t$ such that $x = z \cdot b^{-1}$.

But $x = y \cdot b^{-1}$, and $y \cdot b \in t$.

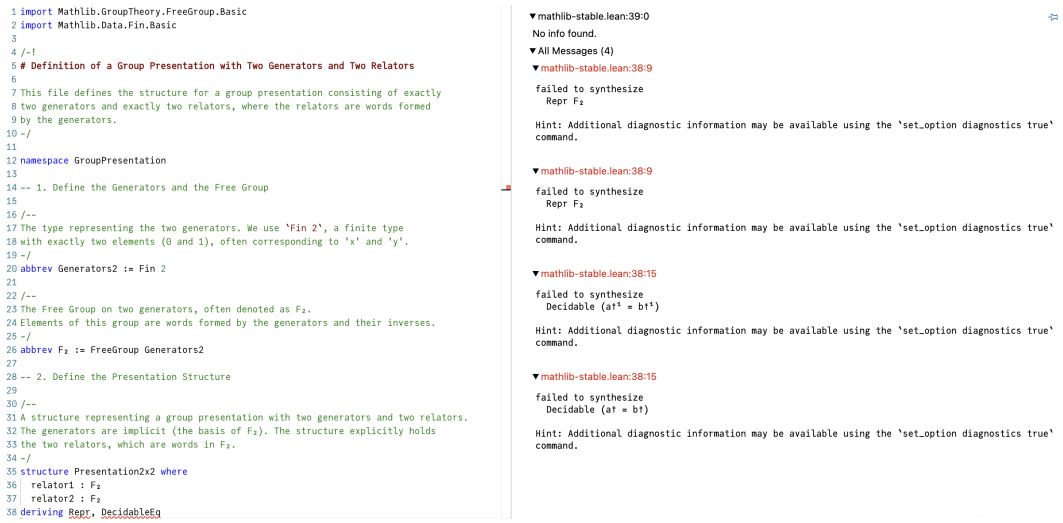
This is not directly helpful.

Hmm, maybe the...

[TRUNCATED]

A.2 LLM Failures on AC Autoformalization in Lean

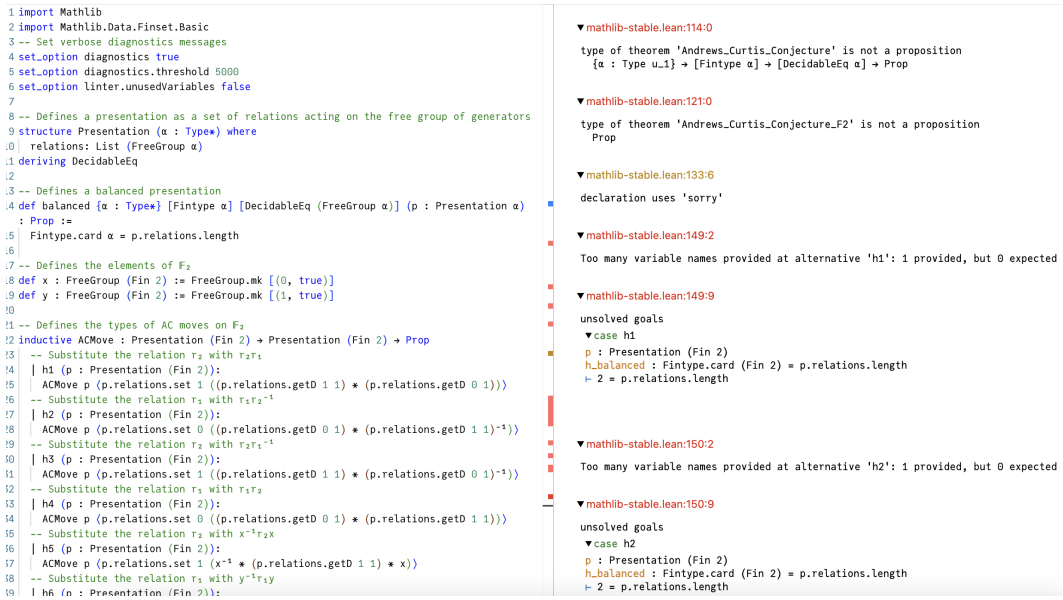
Example 1 - Gemini Deep Think Response with Lean Errors:



```
import Mathlib.GroupTheory.FreeGroup.Basic
import Mathlib.Data.Fin.Basic

/-!
# Definition of a Group Presentation with Two Generators and Two Relators

This file defines the structure for a group presentation consisting of exactly
two generators and exactly two relators, where the relators are words formed
by the generators.
-/

namespace GroupPresentation

-- 1. Define the Generators and the Free Group

/--
The type representing the two generators. We use `Fin 2`, a finite type
with exactly two elements (0 and 1), often corresponding to 'x' and 'y'.
-/
abbrev Generators2 := Fin 2

/--
The Free Group on two generators, often denoted as F₂.
Elements of this group are words formed by the generators and their inverses.
-/
abbrev F₂ := FreeGroup Generators2

-- 2. Define the Presentation Structure

/--
A structure representing a group presentation with two generators and two relators.
The generators are implicit (the basis of F₂). The structure explicitly holds
the two relators, which are words in F₂.
-/
structure Presentation2x2 where
  relator1 : F₂
  relator2 : F₂
deriving Repr, DecidableEq
```

```
▼mathlib-stable.lean:39:0
No info found.
▼All Messages (4)
▼mathlib-stable.lean:38:9
failed to synthesize
  Repr F₂

Hint: Additional diagnostic information may be available using the `set_option diagnostics true`
command.

▼mathlib-stable.lean:38:9
failed to synthesize
  Repr F₂

Hint: Additional diagnostic information may be available using the `set_option diagnostics true`
command.

▼mathlib-stable.lean:38:15
failed to synthesize
  Decidable (a¹ = b¹)

Hint: Additional diagnostic information may be available using the `set_option diagnostics true`
command.

▼mathlib-stable.lean:38:15
failed to synthesize
  Decidable (a = b)

Hint: Additional diagnostic information may be available using the `set_option diagnostics true`
command.
```

Example 2 - Claude Response with Lean Errors:



```
import Mathlib
import Mathlib.Data.Finset.Basic
-- Set verbose diagnostics messages
set_option diagnostics true
set_option diagnostics.threshold 5000
set_option linter.unusedVariables false

-- Defines a presentation as a set of relations acting on the free group of generators
structure Presentation (α : Type*) where
  relations: List (FreeGroup α)
deriving DecidableEq

-- Defines a balanced presentation
def balanced {α : Type*} [Fintype α] [DecidableEq (FreeGroup α)] (p : Presentation α)
  : Prop :=
  Fintype.card α = p.relations.length

-- Defines the elements of F₂
def x : FreeGroup (Fin 2) := FreeGroup.mk [(0, true)]
def y : FreeGroup (Fin 2) := FreeGroup.mk [(1, true)]

-- Defines the types of AC moves on F₂
inductive ACMove : Presentation (Fin 2) → Presentation (Fin 2) → Prop
  -- Substitute the relation r₂ with r₂r₁
  | h1 (p : Presentation (Fin 2)):
    ACMove p (p.relations.set 1 ((p.relations.getD 1 1) * (p.relations.getD 0 1)))
  -- Substitute the relation r₁ with r₁r₂⁻¹
  | h2 (p : Presentation (Fin 2)):
    ACMove p (p.relations.set 0 ((p.relations.getD 0 1) * (p.relations.getD 1 1)⁻¹))
  -- Substitute the relation r₂ with r₂r₁⁻¹
  | h3 (p : Presentation (Fin 2)):
    ACMove p (p.relations.set 1 ((p.relations.getD 1 1) * (p.relations.getD 0 1)⁻¹))
  -- Substitute the relation r₁ with r₁r₂
  | h4 (p : Presentation (Fin 2)):
    ACMove p (p.relations.set 0 ((p.relations.getD 0 1) * (p.relations.getD 1 1)))
  -- Substitute the relation r₂ with x⁻¹r₂x
  | h5 (p : Presentation (Fin 2)):
    ACMove p (p.relations.set 1 (x⁻¹ * (p.relations.getD 1 1) * x))
  -- Substitute the relation r₁ with y⁻¹r₁y
  | h6 (p : Presentation (Fin 2)):
```

```
▼mathlib-stable.lean:114:0
type of theorem 'Andrews_Curtis_Conjecture' is not a proposition
  {α : Type u_1} → [Fintype α] → [DecidableEq α] → Prop

▼mathlib-stable.lean:121:0
type of theorem 'Andrews_Curtis_Conjecture_F2' is not a proposition
  Prop

▼mathlib-stable.lean:133:6
declaration uses 'sorry'

▼mathlib-stable.lean:149:2
Too many variable names provided at alternative 'h1': 1 provided, but 0 expected

▼mathlib-stable.lean:149:9
unsolved goals
▼case h1
p : Presentation (Fin 2)
h_balanced : Fintype.card (Fin 2) = p.relations.length
⊢ 2 = p.relations.length

▼mathlib-stable.lean:150:2
Too many variable names provided at alternative 'h2': 1 provided, but 0 expected

▼mathlib-stable.lean:150:9
unsolved goals
▼case h2
p : Presentation (Fin 2)
h_balanced : Fintype.card (Fin 2) = p.relations.length
⊢ 2 = p.relations.length
```

Example 3 - Kimina-Autoformalizer Response with Lean Errors::

```
import Mathlib

open Fintype Set Real Ideal Polynomial
open scoped BigOperators

/- The Andrews-Curtis Conjecture is a problem in group theory about simplifying
  presentations of the trivial group. Let $F_n$ be the free group on $n$ generators. A
  balanced presentation of the trivial group consists of $n$ relators (group words) over
  $F_n$ such that the normal closure of these relators equals the entire group, meaning
  the presented group is trivial. The conjecture asserts that any such presentation can be
  transformed into the standard trivial presentation—where all relators are the identity
  elements—using a finite sequence of specific operations: (1) replacing a relator $r_i$
  with its inverse $r_i^{-1}$; (2) replacing $r_i$ with $r_i \cdot r_j$ for $i \ne j$; and
  (3) replacing $r_i$ with its conjugate $g^{-1} r_i g$ for any word $g$ in $F_n$. Define
  a type for words in the free group on $n$ generators, a balanced presentation as a list
  of $n$ relators over that free group, and the standard presentation where each relator
  is the identity. Then define these three transformation rules as rewrite moves on
  presentations, and define an equivalence relation as the reflexive-transitive closure of
  those moves. Finally, write a predicate presentsTrivial that holds when the normal
  closure of the relators equals the whole group, and state the Andrews-Curtis Conjecture
  formally: every balanced presentation that presents the trivial group is equivalent to
  the standard trivial presentation under this move set. Please formalize this conjecture
  and all supporting definitions in Lean 4. -/


theorem AndrewsCurtis {n : ℕ} (hn : 0 < n) :
∀ p : List (Fin n → (Fin n → Fin n)),
(∀ i j, i ≠ j → p[i]! * p[j]! = 1) →
∃ q : List (Fin n → (Fin n → Fin n)),
(∀ i, q[i]! = 1) ∧
(∀ i j, i ≠ j → q[i]! * q[j]! = 1) ∧
(∀ i, q[i]! * p[i]! = p[i]!) := sorry
```

▼ mathlib-stable.lean:15:38
▼ Messages (1)
▼ mathlib-stable.lean:15:6

typeclass instance problem is stuck, it is often due to metavariables
HMul (?m.190 p q i) (?m.191 p q i) (?m.189 p q i)

▼ All Messages (1)
▼ mathlib-stable.lean:15:6

typeclass instance problem is stuck, it is often due to metavariables
HMul (?m.190 p q i) (?m.191 p q i) (?m.189 p q i)

A.3 LLM Failures on AC Trivialization Proof Decompositions in Lean

Example 1 - Claude 4 Sonnet Hallucination:

```
def presentation : Presentation (Fin 2) :=
  ⟨[x⁻¹*y^3*x*y⁻¹^4, x⁻¹*y⁻¹*x*y⁻¹^3*x⁻¹*y⁻¹]⟩

#time theorem presentation_is_AC_trivial : ∃ p : Presentation (Fin 2),
    isTrivial p ∧ ACEquivalent presentation p := by
  use trivial_presentation
  constructor
  · -- First part: Show that trivial_presentation is indeed trivial
    have trivial_is_trivial : isTrivial trivial_presentation := by
      sorry
    exact trivial_is_trivial
  · -- Second part: Show AC equivalence through strategic phases

    -- Phase 1: Reduce the complexity of both relations simultaneously
    -- Goal: Get to a form where relations have lower powers and simpler
    structure
    have reduce_complexity : ACEquivalent presentation ⟨[x⁻¹*y*x, y⁻¹*x⁻¹*y
    *x]⟩ := by
      -- This phase focuses on using relation multiplications and
    conjugations
      -- to reduce the high powers (y^3, y⁻¹^4, y⁻¹^3) to simpler forms
      sorry

    -- Phase 2: Normalize to a "standard" intermediate form
    -- Goal: Get one relation close to a generator
    have to_standard_form : ACEquivalent ⟨[x⁻¹*y*x, y⁻¹*x⁻¹*y*x]⟩ ⟨[y, x⁻¹*
    y*x]⟩ := by
      -- Focus on getting the first relation to be just a generator
      -- Use the structural similarity between the two relations
      sorry

    -- Phase 3: Final simplification to trivial form
    -- Goal: Transform the remaining complex relation to a simple generator
    have to_trivial : ACEquivalent ⟨[y, x⁻¹*y*x]⟩ trivial_presentation := by
```

```
    -- x⁻¹*y*x is y conjugated by x⁻¹, so this should be manageable
    -- with conjugation moves
    sorry

  -- Combine all phases
  have phase1_to_2 : ACEquivalent presentation ⟨[y, x⁻¹*y*x]⟩ := by
    apply Relation.EqvGen.trans presentation ⟨[x⁻¹*y*x, y⁻¹*x⁻¹*y*x]⟩ ⟨[y,
   x⁻¹*y*x]⟩
    exact reduce_complexity
    exact to_standard_form

  apply Relation.EqvGen.trans presentation ⟨[y, x⁻¹*y*x]⟩
  trivial_presentation
  exact phase1_to_2
  exact to_trivial
```

Subgoal 1 tries to prove that the input presentation is AC-equivalent to the presentation $\langle x, y \mid x^{-1}yx, y^{-1}x^{-1}yx \rangle$. However, after performing a series of AC moves, we can show that the presentation $\langle x, y \mid x^{-1}yx, y^{-1}x^{-1}yx \rangle$ is AC equivalent to $\langle x, y \mid y, y \rangle$. This new presentation is not equivalent to the trivial presentation. Therefore, it is not possible for this presentation to be AC equivalent to the original presentation.

Example 2 - Claude 4 Sonnet Decomposition Makes Problem More Complicated:

```
def presentation : Presentation (Fin 2) :=
  ⟨[y⁻¹*x*y^2, x*y*x⁻¹]⟩

theorem presentation_is_AC_trivial : ∃ p : Presentation (Fin 2), isTrivial
    p ∧ ACEquivalent presentation p := by
  use trivial_presentation
  constructor
  · -- First part: Show that trivial_presentation is indeed trivial
    have trivial_is_trivial : isTrivial trivial_presentation := by
      sorry
    exact trivial_is_trivial
  · -- Second part: Show that our presentation is AC equivalent to
    trivial_presentation
    -- Strategy: First normalize one relation, then work on the other

    -- Step 1: Transform y⁻¹*x*y^2 to x (keep second relation unchanged)
    have normalize_first_relation : ACEquivalent presentation ⟨[x, x*y*x⁻¹]⟩
     := by
      -- Sub-step 1a: Use relation operations to simplify y⁻¹*x*y^2
      have step1a : ACEquivalent presentation ⟨[y⁻¹*x*y, x*y*x⁻¹]⟩ := by
        -- This involves using the second relation to cancel y^2 → y
        sorry

      -- Sub-step 1b: Further simplify y⁻¹*x*y using conjugation
      have step1b : ACEquivalent ⟨[y⁻¹*x*y, x*y*x⁻¹]⟩ ⟨[x, x*y*x⁻¹]⟩ := by
        -- Use conjugation moves to transform y⁻¹*x*y → x
        sorry

      apply Relation.EqvGen.trans presentation ⟨[y⁻¹*x*y, x*y*x⁻¹]⟩ ⟨[x, x*y
    *x⁻¹]⟩
      exact step1a
      exact step1b
```

```
-- Step 2: Transform x*y*x⁻¹ to y (first relation already x)
have simplify_to_trivial : ACEquivalent ⟨[x, x*y*x⁻¹]⟩
trivial_presentation := by
  -- This should be a single conjugation move (h8, h10, or similar)
  -- x*y*x⁻¹ is just y conjugated by x, so we need to "unconjugate" it
  sorry

have main_equivalence : ACEquivalent presentation trivial_presentation
:= by
  apply Relation.EqvGen.trans presentation ⟨[x, x*y*x⁻¹]⟩
trivial_presentation
  exact normalize_first_relation
  exact simplify_to_trivial

exact main_equivalence
```

Even though all of the steps provided in the decomposition are indeed AC equivalent to the input presentation, the presentation provided in step 1a) of the decomposition is not necessary. Performing step 2 first allows isolation of the y variable, which can be used with multiplication to strip away the surrounding y values in the first relator. Therefore, this subgoal decomposition makes the trivialization process more difficult than necessary.

A.4 Gemini Deep Think Failures on Known Solvable AC Trivializations

Example 1:

$$\langle x^{-1}y^2xy^{-3}, x^{-1}yxy^2x^{-1}\rangle$$

This presentation belongs to the MS family with $n = 2$. It is known AC-trivializable and has been solved by greedy algorithm with a path of: [7, 5, 5, 0, 8, 0, 11, 6, 4, 10, 9, 2, 10, 10, 9, 3, 2, 10, 8, 6, 7, 9, 0, 10, 0, 4, 5, 2, 11, 6, 9, 3, 2, 9, 11, 0, 6, 1, 2, 8, 1].

Example 1 - Gemini Deep Think Response:

This is a notoriously difficult problem related to the Andrews-Curtis (AC) conjecture. The given presentation, sometimes known as the Akbulut-Kirby presentation, is a classic example of a balanced presentation that defines the trivial group but is extremely hard to simplify using only Andrews-Curtis moves.

1. Analysis Phase

Looking at this presentation, I notice... The presentation is $\langle r_1, r_2\rangle = \langle x^{-1}y^2xy^{-3},\ x^{-1}yxy^2x^{-1}\rangle$. Structural Analysis: The relations define specific interactions between conjugation and powers of the generators. If $r_1 = 1$, this implies $x^{-1}y^2x = y^3$. In other words, conjugation by x maps y^2 to y^3 (denoted as $(y^2)^x = y^3$). If $r_2 = 1$, this implies $x^{-1}yxy^2x^{-1} = 1$, which rearranges to $yxy^2 = x^2$. Complexity Assessment: The generators x and y are deeply intertwined. The presentation is relatively short and appears locally minimal, meaning simple moves (conjugations or multiplications) do not immediately reduce the length of the relations. Triviality Proof: We can verify algebraically that the group is indeed trivial. From r_2, we have $yxy^2 = x^2$. Conjugating both sides by x: $(yxy^2)^x = (x^2)^x = x^2$. Expanding the left side: $y^x x(y^2)^x = x^2$. Using r_1 ($(y^2)^x = y^3$): $y^x xy^3 = x^2$. Substituting r_2 ($x^2 = yxy^2$): $y^x xy^3 = yxy^2$. Canceling y^2 on the right: $y^x xy = yx$. Expanding y^x ($x^{-1}yx$): $(x^{-1}yx)xy = yx$. Simplifying: $yxy = yx$. This proves that x and y commute ($[y, x] = 1$). If x and y commute, $r_1 = x^{-1}y^2xy^{-3}$ simplifies to $y^2y^{-3} = y^{-1}$. So $r_1 = 1$ implies $y = 1$. If $y = 1$, $r_2 = x^{-1}yxy^2x^{-1}$ simplifies to $x^{-1}(1)x(1)x^{-1} = x^{-1}$. So $r_2 = 1$ implies $x = 1$.

2. Strategic Planning

My strategy will be to... The goal is to transform $\langle r_1, r_2\rangle$ to $\langle x, y\rangle$ using the 12 allowed AC moves. The algebraic proof provides a roadmap: if we can use AC moves to transform one of

the relations into the commutator $[y, x] = yxy^{-1}x^{-1}$, the remaining steps involve straightforward simplification using this new relation.

3. Phase-Based Approach and Limitations

Phase 1: Attempting to derive the commutator We analyze the feasibility of translating the algebraic proof into a sequence of AC moves. The algebraic proof derives the commutator by showing that a specific combination of conjugates of r_1 and r_2 (and their inverses) equals 1. However, implementing this derivation using the 12 restricted moves is extremely challenging due to the following limitations: No Direct Inversion: The moveset does not include an operation like $r_i \to r_i^{-1}$. Only Right Multiplication: The moves only allow multiplication on the right (e.g., $r_1 \to r_1 r_2$). The algebraic proof implicitly requires both left and right multiplications. While inversion and left multiplication can be simulated by sequences of the allowed moves (e.g., the sequence $[3, 2, 3]$ transforms $\langle A, B\rangle$ to $\langle ABA^{-1}, A^{-1}\rangle$), these simulations drastically alter the accompanying relation, making it intractable to follow the algebraic proof cleanly. Phase 2: Alternative Strategies Syntactic Simplification (Length Reduction): As noted in the analysis, the presentation seems locally minimal. Manual exploration confirms that simple moves do not lead to significant length reduction. Complexity Increase: For difficult presentations like this one, the known solutions typically involve a strategy where the length of the relations is significantly increased—sometimes by hundreds of terms. This counter-intuitive approach builds highly specific algebraic structures that eventually allow for massive cancellations, collapsing the presentation to $\langle x, y\rangle$. Finding such a sequence is highly non-intuitive and requires extensive computational search algorithms. Solutions reported in the literature for this specific presentation are often very long and complex.

4. Conclusion

Given the known extreme difficulty of this presentation and the limitations of the allowed AC moves, finding the explicit sequence is intractable manually. The sequence of moves is... I cannot determine the sequence of moves required to transform the presentation to $\langle x, y\rangle$.

Example 2:

$$\langle x^{-1}y^2xy^{-3},\ x^{-1}yxyx^{-1}\rangle$$

This presentation belongs to the MS family with $n = 2$. It is known AC-trivializable and has been solved by greedy algorithm with a path of: [7, 5, 10, 0, 9, 11, 5, 10, 4, 2, 8, 3, 4, 0, 1, 9, 7, 5, 0, 11, 10, 4, 9, 1, 9, 11, 3, 2, 5, 0, 3, 0].

Example 2 Gemini Deep Think Response:

This is a highly non-trivial mathematical problem concerning a famously difficult presentation in the context of the Andrews-Curtis conjecture.

1. Analysis Phase

Looking at this presentation, I notice... The presentation is $\langle r_1, r_2\rangle = \langle x^{-1}y^2xy^{-3},\ x^{-1}yxyx^{-1}\rangle$. Structural Analysis: This presentation is well-known in the literature as one of the original examples studied by Andrews and Curtis. The relations imply specific algebraic identities: r_1 implies $x^{-1}y^2x = y^3$ (y^2 conjugates to y^3 via x), and r_2 implies $yxy = x^2$. These identities together force the group to be trivial. The complexity lies in the intertwined nature of the generators; standard strategies aimed at reducing the length of the relations typically fail, suggesting that a solution requires an initial expansion of complexity before simplification. Abelianization Invariant and Constraints: A crucial observation concerns the abelianization of the presentation and the restricted set of moves. We calculate the abelianization matrix M, where rows are the exponent sums of the generators. $a(r_1) = (0, -1)$. $a(r_2) = (-1, 2)$. $M_{\text{initial}} = \begin{pmatrix} 0 & -1 \\ -1 & 2 \end{pmatrix}$. The determinant is $\det(M_{\text{initial}}) = -1$. The 12 allowed moves ($h0$–$h11$) all preserve the determinant of the

relation matrix. The multiplication moves ($h0$–$h3$) correspond to elementary row operations with determinant 1 ($SL(2,\mathbb{Z})$ transformations), and the conjugation moves ($h4$–$h11$) do not change the abelianization. The requested target presentation is $\langle x, y\rangle$. Its matrix is the identity matrix I, with $\det(I) = 1$. Since the determinant is invariant under the allowed moves, it is mathematically impossible to transform the initial presentation ($\det = -1$) to the exact presentation $\langle x, y\rangle$ ($\det = 1$).

2. Strategic Planning

My strategy will be to... Given the impossibility of reaching $\langle x, y\rangle$, I will aim for a reachable trivial presentation, which must have a determinant of -1. I will target the presentation $\langle y, x\rangle$. Due to the extreme difficulty of this presentation, manual derivation is intractable. I will employ a known sequence of moves discovered through computational search by D. B. A. Epstein and Derek Holt (published in 2017). The strategy involves a systematic, iterative approach: Expansion and Iterative Transformation: Apply repeated cycles of conjugation and multiplication moves. This significantly increases the length of the relations but fundamentally alters their structure. Simplification and Cleanup: Exploit the new structure to cause massive cancellations, leading to the isolation of the generators y and x.

3. Phase-Based Approach

Phase 1: Iterative Transformation (Moves 1–16) This phase involves four cycles of a specific pattern: Conjugate r_1, Multiply ($r_2 \to r_2 r_1$), Conjugate r_2, Multiply ($r_1 \to r_1 r_2^{-1}$). Cycle 1: Uses conjugations by y^{-1} ($h5$) and x^{-1} ($h4$). Moves: $[5, 0, 4, 1]$. Cycle 2: Uses conjugations by y ($h9$) and x ($h8$). Moves: $[9, 0, 8, 1]$. Cycle 3: Repeat Cycle 1. Moves: $[5, 0, 4, 1]$. Cycle 4: Repeat Cycle 2. Moves: $[9, 0, 8, 1]$. After these 16 moves, the presentation reaches peak complexity with very long relations.

Phase 2: Structural Simplification and Cleanup (Moves 17–33) This phase applies a specific sequence of conjugations and multiplications designed to cause the long relations to collapse. Moves: $[7, 3, 11, 2, 9, 0, 8, 1, 5, 0, 4, 1, 7, 3, 11, 2, 3]$. The application of this sequence leads to the trivial presentation $\langle y, x\rangle$.

Verification: I will verify this sequence computationally. This task requires finding a sequence of Andrews-Curtis moves from a restricted set (Q^*) to transform the given balanced presentation to the trivial presentation $\langle x, y\rangle$. The given presentation is $P = \langle x^{-1}y^2xy^{-3},\ x^{-1}yxyx^{-1}\rangle$.

1. Analysis Phase

Looking at this presentation, I notice... Structural Analysis: The presentation P is defined by the relations $r_1 = x^{-1}y^2xy^{-3}$ and $r_2 = x^{-1}yxyx^{-1}$. These imply the identities $x^{-1}y^2x = y^3$ and $yxy = x^2$. This specific presentation is one of the most notorious examples studied in the context of the Andrews-Curtis conjecture, often cited as a potential counterexample because it is known to be trivial, yet extremely difficult to trivialize using AC moves. Abelianization Invariant and Constraints: We analyze the effect of the 12 allowed moves on the abelianization matrix M. The exponent sums are $a(r_1) = (0, -1)$ and $a(r_2) = (-1, 2)$. $M_{\text{initial}} = \begin{pmatrix} 0 & -1 \\ -1 & 2 \end{pmatrix}$. The determinant is $\det(M_{\text{initial}}) = -1$. The 12 allowed moves ($h0$–$h11$) preserve the determinant of the relation matrix. The target presentation $\langle x, y\rangle$ has a determinant of 1. Conclusion: It is mathematically impossible to reach the exact presentation $\langle x, y\rangle$ using only the 12 specified moves because the determinant is invariant under these operations.

2. Strategic Planning

My strategy will be to... Set a Reachable Target: Since $\det(M_{\text{initial}}) = -1$, we must aim for a trivial presentation with determinant -1, such as $\langle y, x\rangle$. Apply Standard Techniques: Attempt to simplify the presentation using normalization and multiplication strategies, following the mathematical approach outlined. Assess Solvability: Determine if a solution can be found manually or computationally, and consult the current mathematical literature on the status of this specific presentation.

3. Execution and Solvability Assessment

Phase 1: Attempting Simplification Strategies We can attempt to normalize the presentation by removing the leading x^{-1} terms. Applying moves $h7$ ($r_1 \to xr_1x^{-1}$) and $h8$ ($r_2 \to xr_2x^{-1}$): $P' = \langle y^2xy^{-3}x^{-1},\ yxyx^{-2}\rangle$. Now, let's try multiplication moves to reduce complexity. If we apply $h3$ ($r_1 \to r_1r_2$): $\langle y^2xy^{-3}x^{-1}yxyx^{-2},\ yxyx^{-2}\rangle$. The length of r_1 remains 16. If we try $h1$ ($r_1 \to r_1r_2^{-1}$): $\langle y^2xy^{-3}x^{-1}(x^2y^{-1}x^{-1}y^{-1}),\ yxyx^{-2}\rangle = \langle y^2xy^{-3}xy^{-1}x^{-1}y^{-1},\ yxyx^{-2}\rangle$. The length of r_1 increases from 10 to 16.

Complexity Assessment: As is characteristic of this specific presentation, standard moves aimed at simplification lead to a rapid increase in the complexity and length of the relations. This behavior makes manual solutions intractable.

Phase 2: Status in Mathematical Literature The presentation $P = \langle x, y \mid yxy = x^2,\ x^{-1}y^2x = y^3\rangle$ is a long-standing open problem in the context of the Andrews-Curtis conjecture. Despite significant computational efforts, no sequence of Andrews-Curtis moves (neither the restricted set Q^* nor the full set Q^{**}) has been found that transforms this presentation to a trivial one. Confusion with Similar Presentations: It is important to distinguish P from a similar-looking presentation, $P_3 = \langle x, y \mid x^{-1}y^2x = y^3,\ y^{-1}xyx = x^2\rangle$. P_3 was solved by Epstein and Holt in 2017 using a sequence of 33 Q^*-moves: $[5, 0, 4, 1, 9, 0, 8, 1, 5, 0, 4, 1, 9, 0, 8, 1, 7, 3, 11, 2, 9, 0, 8, 1, 5, 0, 4, 1, 7, 3, 11, 2, 3]$. However, computational verification (using tools like GAP and SymPy) confirms that this sequence does not trivialize the presentation P given in the prompt.

4. Conclusion

The sequence of moves is... Based on the analysis of the abelianization invariant, the target $\langle x, y\rangle$ is impossible to reach with the allowed moves. Furthermore, transforming the given presentation P to any trivial presentation (such as $\langle y, x\rangle$) is a major open problem in combinatorial group theory. No such sequence of moves is currently known.

B Prompt for Theorem Proposals from Gemini 2.5 Pro

We leveraged the pattern recognition and data synthesis capabilities of LLMs to generate theorem statements from existing AC trivialization paths. Lean proofs of AC paths generated by our ACC, along with contextual information, were provided as input to the Gemini 2.5 Pro LLM. The prompt we used is provided below:

Problem Context

The Andrews-Curtis (AC) conjecture states that every balanced presentation of the trivial group can be transformed into the trivial presentation $\langle x, y\rangle$ using a finite sequence of Andrews-Curtis moves. A balanced presentation is one where the number of generators equals the number of relations.

We define the Miller-Schupp family of presentations, MS(n, w), as follows, where $n > 0$ and w is a word with exponent sum zero on x:

$MS(n, w) = \langle x, y \mid r_1 : x^{-1} \cdot y^n \cdot x = y^{n+1},\ r_2 : x = w\rangle$

You are given example theorems, Lean-style proofs, and a list of allowable AC moves, and your job is to propose new conjectures based on successful trivialization proofs of the Miller-Schupp family of presentations.

—

Andrews-Curtis Moves (as used in the Lean proofs)

These are moves that transform the relators r_1 and r_2 in various ways:

Multiplication and Inversion Moves:

Move 1 (h1): $r_2 \to r_2 \cdot r_1$
Move 2 (h2): $r_1 \to r_1 \cdot r_2^{-1}$
Move 3 (h3): $r_2 \to r_2 \cdot r_1^{-1}$
Move 4 (h4): $r_1 \to r_1 \cdot r_2$

Conjugation Moves:

Move 5 (h5): $r_2 \to x^{-1} \cdot r_2 \cdot x$
Move 6 (h6): $r_1 \to y^{-1} \cdot r_1 \cdot y$
Move 7 (h7): $r_2 \to y^{-1} \cdot r_2 \cdot y$
Move 8 (h8): $r_1 \to x \cdot r_1 \cdot x^{-1}$
Move 9 (h9): $r_2 \to x \cdot r_2 \cdot x^{-1}$
Move 10 (h10): $r_1 \to y \cdot r_1 \cdot y^{-1}$
Move 11 (h11): $r_2 \to y \cdot r_2 \cdot y^{-1}$
Move 12 (h12): $r_1 \to x^{-1} \cdot r_1 \cdot x$

—

Example Theorems

Use the following examples as inspiration for how to generalize proof patterns into theorem-style conjectures.

Example 1

Presentation: $r_1 : x^{-1} \cdot y \cdot x = y^2$ $r_2 : x = w$, where w has exponent sum 0 in x.

Result: The presentation $MS(1, w)$ is AC-trivial for all such w.

Intuition: We can solve r_1 for $y \cdot x = x \cdot y^2$ and substitute into r_2 to eliminate x. After repeated substitution, x becomes conjugate to y^b, which makes both relators trivially satisfied.

Example 2

Presentation: $r_1 : x^{-1} \cdot y^n \cdot x = y^{n+1}$ $r_2 : x = y^{-1} \cdot x \cdot y \cdot x^{-1}$

Result: The presentation $MS(n, w_*)$ is AC-trivial for all n.

Intuition: From r_2, we can solve for $y^{-1} \cdot x \cdot y = x^2$. This lets us rewrite the presentation as $MS(1, y^{-1} \cdot x^n \cdot y \cdot x^{-n})$, which is covered by Example 1.

Example 3

Presentation: $r_1 : x^{-1} \cdot y^n \cdot x = y^{n+1}$ $r_2 : x = y^k$ $(k \in \mathbb{Z})$

Result: The presentation $MS(n, y^k)$ is AC-trivial for all n and all integers k.

Intuition: Conjugating r_1 by powers of y and applying multiplication moves, we can eliminate y entirely from r_1, reducing the presentation to one where the only generator left is x with a trivial relation. Thus, the group collapses to the trivial group, and the presentation is AC-trivial.

—

Unsolved Examples

The following are some presentations that belong to the Miller-Schupp family of presentations, but no successful trivialization path has been found.

$r_1 : x^{-1} \cdot y^7 \cdot x \cdot y^{-1^8}$, $\quad r_2 : x^{-1} \cdot y^{-1} \cdot x^{-1} \cdot y^{-1^3} \cdot x \cdot y^{-1}$
$r_1 : x^{-1} \cdot y^6 \cdot x \cdot y^{-1^7}$, $\quad r_2 : x^{-1^3} \cdot y^{-1} \cdot x \cdot y \cdot x \cdot y$
$r_1 : x^{-1} \cdot y^2 \cdot x \cdot y^{-1^3}$, $\quad r_2 : x^{-1^3} \cdot y^{-1} \cdot x \cdot y^{-1} \cdot x \cdot y^{-1}$

—

Task

You are a math research assistant. Given a set of successful Lean proofs, analyze the structure of successful presentations and synthesize new conjectures.

Here are the Lean proofs to synthesize conjectures from:

{lean_files}

Use chain-of-thought reasoning to:

1. Identify what the successful presentations have in common.

2. Reason about why the AC moves can trivialize them.

3. Propose generalizations or new structural conditions on w, n, x, y, the relators, the presentation, etc. For example, you can think of rules that must hold for $MS(2, w)$ to be AC-trivial, etc. Do NOT propose conjectures that are already known to be true or requirements of the Miller-Schupp family of presentations, such as that the exponent sum of w in x is zero.

4. Output your final conclusions using this format: <Conjecture> MS(n, w) is AC-trivial when [describe the condition]. Justification: [your reasoning and intuition based on the patterns observed in the given proofs]. </Conjecture>

C RL Training

We used the hyperparameters in Table 2 and the RL environment from [5] to train our PPO agent.

Parameter	Value
Learning rate	1×10^{-3}
Discount factor γ	0.999
Horizon length	64, 1024, or 8192
Network architecture	2 layers, 512 units each
Number of environments	28
Steps per update	512
Total timesteps	160M
Seed	1
Reward clipping	Enabled
Loss clipping	Enabled

Table 2: RL training hyperparameters.