

000 001 002 003 004 005 006 007 008 009 010 011 012 013 014 015 016 017 018 019 020 021 022 023 024 025 026 027 028 029 030 031 032 033 034 035 036 037 038 039 040 041 042 043 044 045 046 047 048 049 050 051 052 053 GAR: GENERATIVE ADVERSARIAL REINFORCEMENT LEARNING FOR FORMAL THEOREM PROVING

Anonymous authors

Paper under double-blind review

ABSTRACT

Solving math problems through verifiable languages such as Lean has significantly impacted both the mathematics and computer science communities. Current state-of-the-art models are often trained with expensive online Reinforcement Learning (RL) or expert iteration. However, these approaches rely on fixed problem sets, which causes inefficient training and limits the model to tackle complex problems. To overcome these limitations, we propose **GAR**: *Generative Adversarial Reinforcement learning*, a comprehensive RL training framework that jointly trains the problem composer and solver in an adversarial loop. **GAR** introduces an implicit curriculum learning mechanism, which aligns task difficulty with the prover’s evolving capability. It thereby improves the training efficiency and enables stronger performance of proving advanced theorems. Experiments show that with **GAR** training, Goedel-Prover-V2-8B and DeepSeek-Prover-V2-7B achieve an average relative improvement in pass@32 of **4.20%** on MiniF2F-Test benchmark, while DeepSeek-Prover-V2’s pass@32 on ProofNet-Test increases from 22.58% to **25.81%**. Beyond formal proving, **GAR** establishes a general RL paradigm for co-evolution of problem generation and solving under verifiable environments.

1 INTRODUCTION

The capability to perform formal mathematical reasoning has long been regarded as both a hallmark of human intelligence and a key objective of machine learning (Newell & Simon, 1956). The ability is typically assessed through rigorous mathematical derivations (Yang et al., 2024b). With the emergence of Large Language Models (LLMs), developing accurate and reliable reasoning has become an active area of research. Recent progress in ZERO RL training (Guo et al., 2025) has further advanced reasoning systems by introducing Long Chain-of-Thought (CoT) thinking models that have self-reflection and self-correction capability.

However, the inherent ambiguity of Natural Language (NL) makes it challenging to verify intermediate reasoning steps. This problem is more severe in advanced mathematics, where the task is to prove theorems rather than give a numerical or formulaic answer. The increasing complexity of modern math compounds this difficulty, as illustrated by the prolonged verification of Fermat’s Last Theorem (Wang et al., 2024). To address this issue, researchers have grounded reasoning in formal logical systems, enabling automatic verification through Formal Language (FL). Based on this idea, some researchers model the reasoning process formally with dependent type languages like Lean (De Moura et al., 2015; Moura & Ullrich, 2021) and Coq (Coq, 1996). Other uses higher-order logic to build language like Isabelle (Paulson, 1994) and HOL (Harrison, 2009). All the above languages make every intermediate step of math reasoning verifiable.

Nevertheless, mastering FLs requires considerable expertise, particularly in dependent-type systems like Lean, where proofs often demand complex type matching and the use of functions with limited data (Wang et al., 2025c). Thus, many works have sought to leverage advances in LLMs to solve FL problems and train specialized FL provers (Polu et al., 2022; Jiang et al., 2022; Xin et al., 2024a; Wang et al., 2024; Lin et al., 2025a; Dong & Ma, 2025b). The verifiability of FLs also motivates large-scale synthesis of new statements, which enables extensive expert iteration (Polu et al., 2022; Xin et al., 2025) or Reinforcement Learning (RL) (Ren et al., 2025; Wang et al., 2025a; Lin et al., 2025b) to further enhance a prover’s performance.

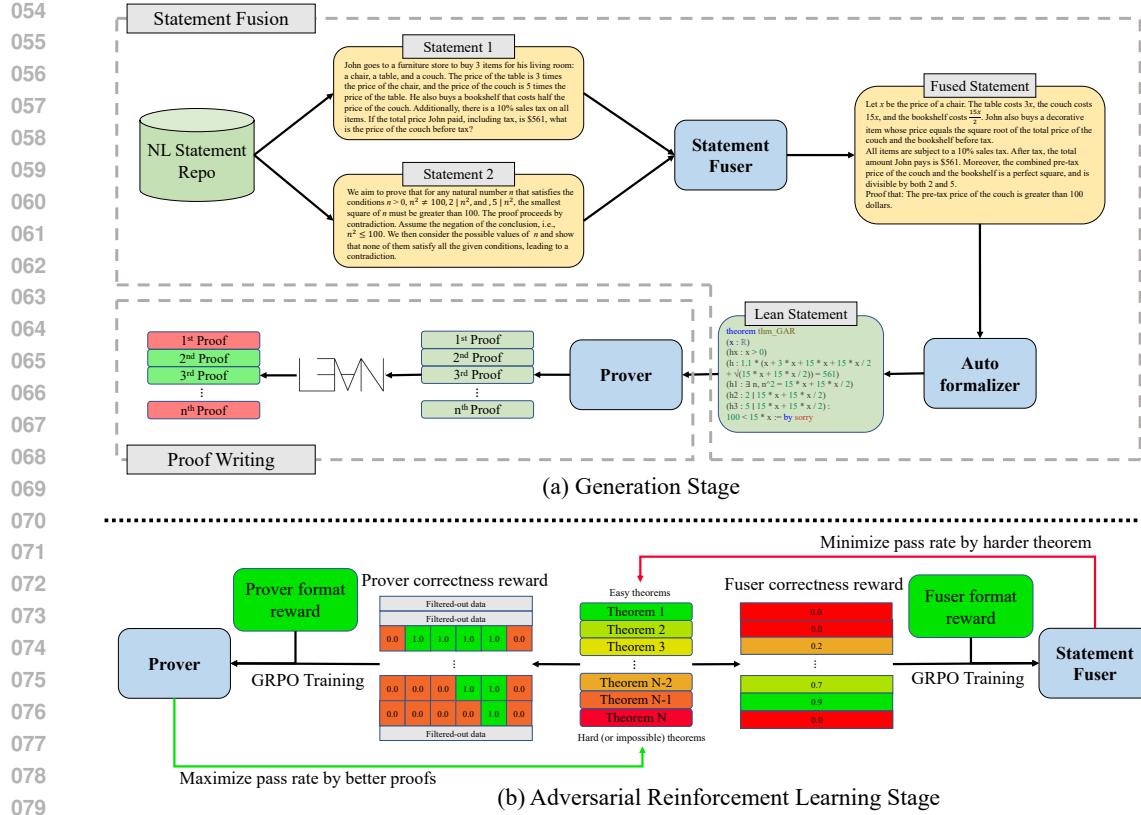


Figure 1: **GAR Training Framework:** Each iteration of **GAR** consists two stages. (a) *Generation Stage*: Pairs of NL statements are sampled from the base repository and combined by the statement fuser to create more challenging problems that fit the current model’s capability. Then, these statements are autoformalized and submitted to the prover to write multiple proofs. Subsequently, the proofs are checked by the Lean verifier for reward assignments. (b) *Adversarial Reinforcement Learning*: The prover is rewarded for producing correct proofs on medium and high-difficulty statements, while the statement fuser is rewarded for generating harder but solvable problems. This adversarial dynamic drives both models to evolve together.

However, most prior approaches in RL or expert iteration rely on fixed theorem statement sets and optimize only the prover. The statement dataset is unchanged during the process. It also leads to wasted computation on trivial or unsolvable tasks. In expert iteration, datasets often require repeated annotation. On the other hand, advanced RL methods suffer from the absence of an adaptive difficulty level of statements during rollouts, which restricts concentrated exploration and limits progress on complex theorems. More broadly, only a few works discuss a systematic way to align problem difficulty with the prover’s growing skill.

To address this limitation, we propose **GAR**: *Generative Adversarial Reinforcement Learning*, a comprehensive RL training framework as presented in Figure 1. **GAR** jointly optimizes the prover and the problem composer, termed as the statement fuser, through adversarial training. This process establishes an implicit curriculum learning that dynamically adjusts statement difficulty to match the prover’s development. Each iteration of the **GAR** framework consists of two stages: the generation and the adversarial RL stage. In the generation stage, the statement fuser synthesizes harder statements from existing solvable ones, and the prover attempts to solve them. In the RL training stage, the fuser is rewarded for producing difficult yet solvable statements, while the prover is rewarded for correctly proving medium and high-difficulty problems. This adversarial setup ensures that statement complexity evolves alongside the prover’s capabilities, resulting in more efficient and effective training.

108 We summarize our contributions as follows: (1) We introduce **GAR**, a comprehensive RL training
 109 framework that establishes implicit curriculum learning, improving the prover’s reasoning ability
 110 while preventing wasted effort on trivial or unsolvable statements. (2) We present *Statement Fusion*
 111 technique, which enables the creation of novel formal statements beyond direct formalization of NL
 112 problems, producing theorem statements better aligned with model capability. (3) We demonstrate
 113 the effectiveness of the **GAR** through extensive experiments by training two base models using
 114 **GAR**. We achieve an average of **4.20%** relative improvement on MiniF2F-Test (Zheng et al., 2021)
 115 and raise DeepSeek-Prover-8B’s pass@32 on the more challenging ProofNet-Test (Azerbayev et al.,
 116 2023) from 22.58% to **25.81%**. It highlights the effectiveness and generality of **GAR**.

117 Furthermore, the **GAR** contributes a general RL paradigm for the co-evolution of problem genera-
 118 tion and solving in verifiable environments, which offers new directions for adversarial co-training
 119 in other reasoning-intensive domains. To facilitate the advancement of the field, we will open-source
 120 the training and inference code of **GAR** in the near future.

121

122 2 METHODOLOGY

123

124 In this section, we introduce the **GAR** framework in detail. The prover’s task is to produce Lean4
 125 proofs from NL-FL statement pairs. Our framework trains the prover to improve at the task by
 126 adapting theorem difficulty to the prover’s evolving capabilities through adversarial training of both
 127 the prover and the statement fuser. Each iteration of **GAR** consists of two stages: the *Generation*
 128 *stage* (Section 2.1) and the *Adversarial Reinforcement Learning stage* (Section 2.2). We first de-
 129 scribe the behavior of each stage in a single iteration in detail, then present the complete procedure
 130 in Section 2.3.

131

132 2.1 GENERATION STAGE

133

134 The generation stage of **GAR** consists of two processes, namely statement fusion, which produces
 135 new statements, and proof writing, which generates and evaluates candidate proofs.

136

137 2.1.1 STATEMENT FUSION

138

139 This process generates harder statements from existing ones. It aims to mitigate the mismatch be-
 140 tween the fixed datasets and the prover’s evolving ability. This process first samples two NL state-
 141 ments from the base dataset, formally: $s_{base} = (s_1^{(NL)}, s_2^{(NL)}) \sim \mathcal{D}_{stat}$, where \mathcal{D}_{stat} is a repository
 142 consists of 793,243 NL statements collect from Lean-Workbook (Wu et al., 2024b) and Numin-
 143 Math (LI et al., 2024). The sampled pair is then passed to the statement fuser trained in the previous
 144 iteration of **GAR**. The fuser is instructed to synthesize a more challenging statement by combining
 the key ideas of the two inputs, namely:

145

$$s^{(NL)} = Fuser_{i-1}(s_{base})$$

146

147 where i denotes the current iteration index, $s^{(NL)}$ is the generated NL statement, and $Fuser_{i-1}$ is
 148 the fuser from the prior step (initialized as base model when $i = 0$).

149

150 In particular, we chose thinking models like Qwen3 (Team, 2025) or DeepSeek-R1 (Guo et al.,
 151 2025) as the base model for the statement fuser because of its outstanding performance. However,
 152 direct use of their native Long CoT capability often results in overthinking and degraded quality
 153 of the generated statement. To mitigate this, we reinitialize the thinking process by passing the
 154 default thinking stage and restarting it with a dedicated indicator token named `<analysis>`. This
 155 trick yields more focused and task-specific reasoning. The full prompt for the statement fusion is
 156 shown in Figure 2. Subsequently, the generated NL statement is passed to the autoformalizer, which
 157 converts the NL statement into Lean statement $s^{(FL)}$. The formalized statement is then passed to
 Lean4 for compilation check. This step filters out statements with grammatical errors.

158

159 In the statement fusion stage, they deliberately separate the fusion of NL statements from the formal-
 160 ization of FL statements. Two key insights guide this design. First, prior studies (Wang et al., 2024;
 161 Xin et al., 2024b) have shown that NL statements play a crucial role in supporting formal reasoning.
 Secondly, current 8B-scale general LLMs exhibit only a limited understanding of FL. This causes
 them to fail to capture essential relations, which hinders the generation of more robust statements.

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

You are an expert mathematics educator skilled in problem design. Your task is to combine multiple given problems into a SINGLE, more challenging problem by combining their key elements. Follow these steps: Please first do the following steps as your analysis process enclosed within `<analysis></analysis>`.

1. Analyze the points of knowledge that need to be used when solving the proof problem and identify overlapping or complementary aspects (e.g., shared topic areas or contrasting difficulty levels)

2. Draft the new problem that integrates at least 2 key components from each original problem and make sure the new problem requires multi-step reasoning (e.g., combining algebraic manipulation with probabilistic analysis). Also, your combined problem should have no non-trivial extension.

3. Additionally, you should make sure that the new problem is solvable.

After your analysis, you should put the new problem into an MD code block. The new problem should be a SINGLE proof problem.

Here are the statements you need to fuse:

Problem 1: `<statement_1>`

Problem 2: `<statement_2>`

Figure 2: Prompt for the Statement Fuser to generate harder statements from existing ones and restart thinking using a new indicator in **GAR** generation stage

Consequently, the fuser produces statements that have a high likelihood of failing the compilation check if we directly fuse formal statements, as seen in Dong & Ma (2025b). Thus, **GAR** firstly fuses the statement in NL and formalizes it to FL, ensuring the fused statements are adaptively more challenging than the base problems to fit prover performance evolution.

2.1.2 PROOF WRITING

Utilizing the NL-FL statement pair generated by the fusion process, the prover from the last iteration generates n candidate proofs (with $n = 16$ in our implementation), denoted as $\{o_k\}_{k=1}^n$, specifically:

$$o_k = \text{Prover}_{i-1}(s^{(NL)}, s^{(FL)}), \forall k \in [1, n]$$

When $i = 0$, it is the base prover model like deepSeek-Prover-V2 (Ren et al., 2025) or Goedel-Prover-V2 (Lin et al., 2025b).

The set of candidate proofs $\{o_k\}_{k=1}^n$ is then passed to the Lean4 verifier to check correctness. Based on the verification results, we compute the empirical pass rate p of this statement for reward assignment as $p = \frac{n_{pass}}{n}$, where n_{pass} is the number of proofs that pass verification.

2.2 ADVERSARIAL REINFORCEMENT LEARNING

This section outlines the training process of **GAR**, where the prover and statement fuser are optimized in competition, deriving adversarial learning and mutual improvement.

2.2.1 STATEMENT FUSER TRAINING

The statement fuser is trained to synthesize problems that are slightly beyond the prover's current capability, thereby creating an implicit curriculum. To achieve this, we assign rewards to the generated NL statements that encourage the production of more challenging but solvable problems. We adopt a variant of Group Relative Policy Optimization (GRPO) (Shao et al., 2024) for this optimization. Specifically, for each step i , the fuser generates N statements (with $N = 1024$ in our implementation). Formally, generated statement set is: $\{s_{i,j}^{(NL)}\}_{j=1}^N$, and base statement set is represented as:

$\{s_{i,j}^{(base)} = (s_{i,j,1}^{(NL)}, s_{i,j,2}^{(NL)})\}_{j=1}^N$. The statement fuser $Fuser_{i-1}$ is then updated by maximizing the following objective.

$$\begin{aligned}
216 \\
217 \quad \mathcal{J}_{GRPO}^{(F)}(\theta) = & \mathbb{E} \left[\left\{ \mathbf{s}_{i,j}^{(base)} \right\}_{j=1}^N \sim \mathcal{D}_{stat}; \left\{ s_{i,j}^{(NL)} \sim \pi_{\theta_{old}} \left(\mathbf{s}_{i,j}^{(base)} \right) \right\} \right] \\
218 \quad & \frac{1}{N} \sum_{j=1}^N \left(\min \left\{ \delta_{i,j}^{(stat)} A_{i,j}^{(stat)}, \text{clip} \left\{ \delta_{i,j}^{(stat)}, 1 \pm \varepsilon \right\} A_{i,j}^{(stat)} \right\} - \beta \mathbb{D}_{KL}(\pi_{\theta} \parallel \pi_{\theta_{ref}}) \right), \\
219 \quad & (1)
\end{aligned}$$

$$\delta_{i,j}^{(stat)} = \frac{\pi_{\theta} \left(s_{i,j}^{(NL)} \mid \mathbf{s}_{i,j}^{(base)} \right)}{\pi_{\theta_{old}} \left(s_{i,j}^{(NL)} \mid \mathbf{s}_{i,j}^{(base)} \right)}, \quad (2)$$

$$\mathbb{D}_{KL}(\pi_{\theta} \parallel \pi_{\theta_{ref}}) = \frac{\pi_{\theta_{ref}} \left(s_{i,j}^{(NL)} \mid \mathbf{s}_{i,j}^{(base)} \right)}{\pi_{\theta} \left(s_{i,j}^{(NL)} \mid \mathbf{s}_{i,j}^{(base)} \right)} - \log \frac{\pi_{\theta_{ref}} \left(s_{i,j}^{(NL)} \mid \mathbf{s}_{i,j}^{(base)} \right)}{\pi_{\theta} \left(s_{i,j}^{(NL)} \mid \mathbf{s}_{i,j}^{(base)} \right)} - 1 \quad (3)$$

230 where π_{θ} denotes the policy model with parameter θ . It represents the statement fuser here. θ_{ref}
231 is the parameter for base statement fuser, θ_{old} is the parameter for the fuser in the previous step,
232 ε and β are hyper-parameters, and $A_{i,j}$ is the advantage, computed from the reward set $\mathbf{r}_i^{(stat)} =$
233 $\{r_{i,j}^{(stat)}\}_{j=1}^{N_i}$ by:

$$A_{i,j}^{(stat)} = \frac{r_{i,j}^{(stat)} - \text{mean}(\mathbf{r}_i^{(stat)})}{\text{std}(\mathbf{r}_i^{(stat)})}, \quad r_{i,j}^{(stat)} = (1 - p_{i,j}) \cdot (1 - m_{i,j}) \cdot \mathbb{I}\{p_{i,j} \neq 0\} \quad (4)$$

234 where $p_{i,j}$ is the prover's empirical pass rate on $s_{i,j}^{(FL)}$, and $m_{i,j}$ is the statement modification rate,
235 which indicates the portion of proofs that the prover tries to modify its statement. The reward is
236 set to 0 if the prover fails to solve the problem, which indicates the statement is too difficult or
237 unsolvable.

238 We introduce the term $1 - m_{i,j}$ as a soft statement modification penalty to balance the risk between
239 reward hacking and the need to preserve model capability. Because of Long CoT training, current
240 expert provers often acquire strong self-correction capability. While valuable, this ability can lead
241 the model to change formal statements during proof writing. This may lead to serious reward hacking
242 if unconstrained. Conversely, a strict ban on such modifications would suppress self-correction
243 and reduce proofreading accuracy. Our soft penalty discourages excessive statement change without
244 too much harm to the result.

245 In summary, the training design for the statement fuser rewards the model to lower the prover's pass
246 rate by composing more challenging statements.

2.2.2 PROVER TRAINING

247 To achieve adversarial training, the prover is optimized to maximize the pass rate on generated state-
248 ments. We employ a variant of the GRPO algorithm for such training. Specifically, for each theorem
249 statement $\mathbf{s}_{i,j} = (s_{i,j}^{(NL)}, s_{i,j}^{(FL)})$, we update the prover model $Prover_{i-1}$ by maximizing the fol-
250 lowing objective function:

$$\begin{aligned}
251 \quad \mathcal{J}_{GRPO}^{(P)}(\omega) = & \mathbb{E} \left[\left\{ o_{i,j,k} \right\}_{k=1}^n \sim \pi_{\omega_{old}}(\mathbf{s}_{i,j}) \right] \\
252 \quad & \frac{1}{n} \sum_{k=1}^n \left(\min \left\{ \delta_{i,j,k}^{(pr)} A_{i,j,k}^{(pr)}, \text{clip} \left\{ \delta_{i,j,k}^{(pr)}, 1 \pm \varepsilon \right\} A_{i,j,k}^{(pr)} \right\} - \beta \mathbb{D}_{KL}(\pi_{\omega} \parallel \pi_{\omega_{ref}}) \right), \\
253 \quad & (5)
\end{aligned}$$

$$\delta_{i,j,k}^{(proof)} = \frac{\pi_{\omega} (o_{i,j,k} \mid \mathbf{s}_{i,j})}{\pi_{\omega_{ref}} (o_{i,j,k} \mid \mathbf{s}_{i,j})} \quad (6)$$

254 where π_{ω} is the prover as policy model, ω is the parameter of prover, and $A_{i,j,k}$ is the advantage of
255 proof $o_{i,j,k}$, computed from the reward set $\mathbf{r}_{i,j}^{(proof)} = \{r_{i,j,k}^{(proof)}\}_{k=1}^n$ by:

$$A_{i,j,k}^{(proof)} = \frac{r_{i,j,k}^{(proof)} - \text{mean}(\mathbf{r}_{i,j}^{(proof)})}{\text{std}(\mathbf{r}_{i,j}^{(proof)})}, \quad r_{i,j,k} = 1 - 0.5 \cdot m_{i,j,k} \quad (7)$$

270 where $m_{i,j,k}$ is a binary indicator of whether a statement modification occurred in proof $o_{i,j,k}$.
 271 Similar to the training of the statement fuser, modifications are penalized but not strictly prohibited.
 272 Furthermore, to ensure the prover is trained on high-quality data, we exclude statements with an
 273 empirical pass rate of 0 (unsolvable) or above 0.5 (too easy) following experience in Wang et al.
 274 (2025a); Dong & Ma (2025b). It make sure the prover is only trained with hard and medium-
 275 level problems. This optimization scheme drives the prover to compete with the statement fuser by
 276 continually enhancing its proof-generation capability.

278 2.3 SUMMARY

280 The **GAR** is an iterative framework that loops the generation stage and the RL stage details above.
 281 To provide a clear overview, we present the complete framework in the form of pseudo-code as
 282 follows:

283 **Algorithm 1 GAR**

284 **Require:** \mathcal{D}_{stat} , Statement Fuser (base) as $Fuser$, Prover (base) as $Prover$, Autoformalizer as AF
 285
 286 1: $\pi_\omega, \pi_{\omega_{old}}, \pi_{\omega_{ref}} = Prover; \pi_\theta, \pi_{\theta_{old}}, \pi_{\theta_{ref}} = Fuser$
 287 2: **for** $i : [1, T]$ **do** ▷ Step i of **GAR**
 288 3: $\{\mathbf{s}_{i,j}^{(base)}\}_{j=1}^N \sim \mathcal{D}_{stat}$ ▷ Sample base statements
 289 4: $\{\mathbf{s}_{i,j}^{(NL)} \sim \pi_\theta(\mathbf{s}_{i,j}^{(base)})\}_{j=1}^N$ ▷ Fuse NL statements
 290 5: $\{\mathbf{s}_{i,j}^{(FL)} \sim AF(s_{i,j}^{(NL)})\}_{j=1}^N$ ▷ Autoformalize statement
 291 6: $\{\mathbf{s}_{i,j} = (s_{i,j}^{(NL)}, s_{i,j}^{(FL)})\}_{j=1}^N$
 292 7: **for** $j : [1, N]$ **do** ▷ Generate proofs
 293 8: $\{o_{i,j,k} \sim \pi_\omega(\mathbf{s}_{i,j})\}$
 294 9: Obtain $p_{i,j}, m_{i,j}$ by Lean checking of proofs
 295 10: $\pi_\theta = \pi_{\theta_{old}} \leftarrow Optimize(\mathcal{J}_{GRPO}^{(F)}(\theta))$ ▷ Train NL Fuser
 296 11: $\pi_\omega = \pi_{\omega_{old}} \leftarrow Optimize(\mathcal{J}_{GRPO}^{(P)}(\omega))$ ▷ Train Prover
 297
 298 12: **return** π_θ, π_ω

300 When trained in multiple iterations, the **GAR** establishes an implicit curriculum by aligning state-
 301 ment difficulty with the prover’s evolving capability. The statement fuser is trained to reduce the
 302 pass rate by generating more challenging statements, while the prover is optimized to increase it by
 303 producing more valid proofs. Together, their adversarial interaction drives progressive improvement.

305 3 EXPERIMENTS

306 We conduct comprehensive experiments on the MiniF2F-Test (Zheng et al., 2021) and ProofNet-
 307 Test (Azerbayev et al., 2023) benchmarks to assess the performance of the **GAR** framework in for-
 308 mal proof writing. Specifically, we show in Section 3.3 that the models trained with **GAR** achieve
 309 better empirical results, confirm in Section 3.4 that adversarial training induces an implicit curricu-
 310 lum by generating progressively harder statements, and report ablation study results in Section 3.5.
 311 Due to space limitations, we have included the efficiency study and case study in Appendix C.

314 3.1 EXPERIMENT SETUP

316 3.1.1 DATASET AND TASK

317 We measure the Lean4 reasoning capability of the **GAR** trained model by MiniF2F-Test (Zheng
 318 et al., 2021) and ProfNet-Test (Azerbayev et al., 2023) benchmarks. They are challenging bench-
 319 marks and adopted in nearly all major studies in the field (Xin et al., 2024b; Lin et al., 2024; Wang
 320 et al., 2024; Wu et al., 2024a; Polu et al., 2022; Dong & Ma, 2025a; Lin et al., 2025b; Wang et al.,
 321 2025c).

322 The MiniF2F-Test benchmarks comprise 244 Lean4 statements, spanning from high school com-
 323 petition problems to elementary undergraduate-level theorem proofs. Specifically, MiniF2F-Test

324

325

Table 1: Main experimental results of **GAR** trained models compared to recent provers models.

Method	Sample budget	MiniF2F-Test	ProofNet-Test
Lean-STaR (Lin et al., 2024)	$64 \times 1 \times 50$	46.31%	-
InternLM-2.5-StepProver (Wu et al., 2024a)	$4 \times 32 \times 600$	50.70%	18.80%
DeepSeek-Prover-V1.5-RL (Xin et al., 2024b)	128	50.00%	18.20%
STP-Lean (Dong & Ma, 2025b)	128	56.15%	19.50%
MA-LoT (Wang et al., 2025c)	32	61.07%	15.47%
Kimina-Prover-Distill-7B (Wang et al., 2025a)	32	63.10%	-
DeepSeek-Prover-V2-7B (Ren et al., 2025)	32	70.49%	22.58%
Geodel-Prover-V2-8B (Lin et al., 2025a)	32	77.87%	-
<i>Our models</i>			
GAR on Deepseek-Prover-V2	32	74.18%	25.81%
GAR on Goedel-Prover-V2	32	80.33%	-

335

Table 2: Average proof correctness rate for Goedel-Prover-V2-8B (base model) and GAR model trained on base model.

Step idx	Base Model	GAR model
0	29.16%	29.16%
1	16.50%	23.71%
2	11.58%	20.53%
3	7.61%	20.08%
4	7.69%	21.79%

Table 3: Statement modification rate for dropping statement modification penalty and full GAR trained models.

Step idx	w/o Stat. Penalty	Full GAR
0	42.94%	42.94%
1	48.18%	48.72%
2	60.42%	30.50%
3	71.82%	39.65%
4	74.11%	33.63%

348

comprises problems formalized from the MATH dataset (Hendrycks et al., 2021), high school competitions such as AMC, AIME, and IMO, as well as self-crafted problems. ProofNet-Test consists of 186 theorems formalized from standard undergraduate textbooks on advanced topics such as real and complex analysis, linear algebra, abstract algebra, and topology. In our setting, we train the LLM with **GAR** to generate Lean4 proofs from the NL-FL statement pair. To avoid overloading the model, all the imports and namespaces are manually configured.

354

3.1.2 BASELINES

356

To evaluate the effectiveness of **GAR**, we compare it against strong open-source baselines, including Lean-STaR (Lin et al., 2024), InternLM-2.5-StepProver (Wu et al., 2024a), Kimina-Prover-Preview-Distill-7B (Wang et al., 2025a), DeepSeek-Prover-V1.5-RL (Xin et al., 2024b), STP-Lean (Dong & Ma, 2025a), MA-LoT (Wang et al., 2025c), Goedel-Prover-V2 (Lin et al., 2025b), and DeepSeek-Prover-V2 (Ren et al., 2025). For baseline models that require Long CoT reasoning (Kimina-Prover, MA-LoT, DeepSeek-Prover-V2, and Goedel-Prover-V2), we restrict the reasoning length to 16,384 tokens to conserve computational resources.¹

363

3.2 IMPLEMENTATION DETAILS

365

In the generation stage, we construct a repository of 793,243 NL statements from Numina-Math (LI et al., 2024) and Lean-Workbook (Ying et al., 2024) datasets. For statement fusion, we employ Qwen3-8B (Team, 2025) as the base model for the statement fuser because of its skip-thinking capability. We apply Kimina-Autoformalizer-7B (Wang et al., 2025a) as the autoformalizer. For proof generation, we use DeepSeek-Prover-V2-7B (Ren et al., 2025) and Goedel-Prover-V2 (Lin et al., 2025b) as base models for provers. We sample 1,024 theorems per step and generate 16 proofs per theorem following (Wang et al., 2025a). We also restrict the sequence length to 16,384 tokens for both models. The GRPO hyperparameters are set with a learning rate of 2E-6, $\varepsilon = 0.2$, and $\beta = 0.01$. We perform three iterations of **GAR** training on Goedel-Prover-V2 and five iterations on DeepSeek-Prover-V2, which costs around 140 H100 hours for each training. On average, DeepSeek-Prover-V2 costs fewer hours per iteration due to average shorter thinking. In verification, any proof

377

¹Because of this restriction, our reported results for DeepSeek-Prover-V2 and Goedel-Prover-V2 differ from those in the original papers, where the evaluations used a context length of 40,960.

378 involving statement modification or relying on the “sorry” or “admit” tactic is counted as incorrect
 379 to ensure fairness.
 380

381 3.3 RESULTS 382

383 Table 1 demonstrates the empirical results of applying **GAR** to train base prover models. For
 384 Goedel-Prover-V2, MiniF2F-Test pass@32 raises to 80.33%, indicating a **3.16%** relative gain. For
 385 DeepSeek-Prover-V2, the MiniF2F-Test score improves to 74.18%, corresponding to a **5.23%** in-
 386 crease. On the more challenging ProofNet-Test benchmark, which targets advanced mathematics
 387 topics, **GAR** enhances DeepSeek-Prover-V2’s pass@32 rate from 22.58% to **25.81%**.²

388 Compared with other baselines, both the base model and the **GAR**-trained models achieve consistent
 389 and substantial gains, highlighting that **GAR** can contribute to models that already have outstanding
 390 performance. Compared to base models, the improvements suggest that the adversarial method pro-
 391 posed in **GAR** enables provers to tackle increasingly difficult problems by establishing an implicit
 392 curriculum. Such a curriculum can progressively adapt problem difficulty to the model’s capability,
 393 allowing the prover to explore deeper reasoning strategies and solve more advanced theorems. This
 394 behavior becomes more significant as the theorems get harder.

395 3.4 STATEMENT FUSER STUDY 396

397 To show that **GAR** produces increasingly more difficult statements across iterations and thereby
 398 establishes an implicit curriculum learning, we examine the difficulty of problems generated by the
 399 statement fuser. In each iteration, we randomly sample 50 generated statements and compute their
 400 average proof correctness rate using both the base prover and **GAR**-trained prover at that iteration.
 401 Details of the metric we used are provided in Appendix A.2. The result of this experiment is reported
 402 in Table 2.

403 From the results, we can observe that the base model exhibits a consistent performance degradation,
 404 with accuracy falling from 29.16% at the first iteration’s data to 7.69% by the fifth. In contrast, the
 405 **GAR**-trained model shows only a minor initial decline due to the statement matching but stabilizes at
 406 around 21% across later iterations. These findings confirm that **GAR** progressively generates harder
 407 statements, as evidenced by the base model’s decline. The maintained trained model’s performance
 408 demonstrates the adversarial training strength of the prover’s performance over time.

409 3.5 ABLATION STUDIES 410

411 3.5.1 EFFECT OF STATEMENT MODIFICATION PENALTY 412

413 We evaluate the impact of the statement modification penalty by monitoring statement mod-
 414 ification rates across by the provers during training rounds. Details of this metric are pro-
 415 vided in Appendix A.3. We trained a variant of **GAR** on Goedel-Prover-V2-8B without
 416 the penalty in both the statement fuser and prover, and compared it with the full **GAR**.
 417 Table 3 shows that without the modification penalty, the
 418 prover exploits its self-correction ability by simplifying
 419 statements. Such behavior worsens as the training pro-
 420 gresses; by the fourth step, 74% of statements were mod-
 421 ified at least once, which is a clear signal of reward hack-
 422 ing. In contrast, with the penalty enabled, the mod-
 423 ification rate remains below 40% throughout. This study
 424 confirms the effectiveness of our statement penalty in pre-
 425 venting reward hacking.

426 3.5.2 COMPARE TO DIRECT RL TRAINING 427

428 This experiment tests whether jointly evolving the prover and statement fuser is more effective than
 429 training the prover alone using existing data. We conducted three additional GRPO iterations on

430 Table 4: Pass@32 results on MiniF2F-
 431 Test for Goedel-Prover-V2-8B and its
 432 GRPO and GAR trained version.

Method	MiniF2F-Test
Base model	77.87%
GRPO trained	77.46%
GAR trained	80.33%

433 ²We do not report Goedel-Prover-V2’s accuracy on ProofNet due to the absence of the reference results
 434 in Lin et al. (2025b).

432 Goedel-Prover-V2-8B using the same sampling budget and formalized NL base dataset as **GAR**.
 433 The results are shown in Table 4. The **GAR**-tained model outperforms the variant trained with
 434 traditional GRPO. We conclude that such an improvement to **GAR** progressively raises problem
 435 difficulty, enabling the prover to handle more complex statements. In contrast, further RL training on
 436 static dataset degrades performance, as the base model is already heavily RL trained. These findings
 437 further support the generality of **GAR**, demonstrating its ability to enhance model’s performance
 438 while standard RL no longer provides benefits.

4 RELATED WORK

4.1 LLM FOR FORMAL THEOREM PROVING

444 Recently, applying LLMs to support formal theorem proving has become a prominent research
 445 direction. Training approaches can be broadly divided into two categories: model trained solely
 446 with Supervised Fine-Tuning (SFT) and those combining SFT with Reinforcement Learning (RL).
 447 Early SFT-based provers include Expert Iteration (Polu & Sutskever, 2020), Re-Prover (Yang et al.,
 448 2024c), TheoremLlama (Wang et al., 2024), DeepSeek-Prover-V1 (Xin et al., 2024a), InternLM-
 449 2.5-StepProver (Wu et al., 2024a), MA-LoT (Wang et al., 2025c), and Goedel-Prover-V1 (Lin et al.,
 450 2025a). These models typically require multiple rounds of large-scale annotation with existing for-
 451 mal solvers, which demands a significant amount of computational resources and limits exploration.
 452 To further advance the prover, researchers began incorporating advanced RL techniques. For in-
 453 stance, DeepSeek-Prover-V1.5 (Xin et al., 2024b) employs DPO. After the “ZERO” RL technique
 454 that enables Long CoT thinking developed by Guo et al. (2025), provers like Kimina-Prover (Wang
 455 et al., 2025a), DeepSeek-Prover-V2 (Ren et al., 2025), and Goedel-Prover-V2 (Lin et al., 2025b)
 456 enable models to produce better formal proof after thinking. However, existing RL approaches still
 457 rely on a fixed collection of statements, preventing statements from adapting to the prover’s evol-
 458 ing skill. In contrast, **GAR** jointly trains a statement fuser and a prover, ensuring that generated
 459 theorems remain suitably challenging as the model improves.

4.2 RL METHODS FOR LLM

462 RL has been central to the development of reasoning LLMs (Xiong et al., 2025). Early efforts such
 463 as DeepSeek-Math (Shao et al., 2024) and Qwen-2.5-Math (Yang et al., 2024a) applied reward mod-
 464 eling with GRPO algorithm to enhance exploration. The release of OpenAI-O1 (OpenAI, 2024) and
 465 DeepSeek-R1 (Guo et al., 2025) demonstrated that verifier-based rewards in RL can enable complex
 466 reasoning strategies, including backward search and self-correction. Such techniques have been
 467 widely adopted in systems like Qwen3 (Team, 2025) and Kimi-K2 (Team et al., 2025). Neverthe-
 468 less, nearly all prior work uses the verification signal only to optimize the problem solver, leaving
 469 the problem composer untrained. On the other hand, **GAR** simultaneously improves both statement
 470 fuser and the prover, creating an implicit curriculum that adapts task difficulty and allows the model
 471 to acquire more complex reasoning skills step-by-step.

5 CONCLUSION

474 This paper presents **GAR**, a comprehensive training framework for formal theorem proving. **GAR**
 475 aims to resolve the inefficient and suboptimal performance caused by traditional expert iteration and
 476 online RL in prover training. **GAR** achieves more efficient training by using statement fusion to
 477 formulate statements in the RL process to avoid the prover from annotating problems beyond its
 478 capability range. After **GAR** training, provers can solve more advanced problems through the im-
 479 plicit curriculum learning that enables the model to explore further. Furthermore, **GAR** can improve
 480 the performance of models that have been through heavy RL training by letting the model explore
 481 more difficult statements during the training process. Experiments of applying **GAR** to DeepSeek-
 482 Prover-V2 and Goedel-Prover-V2 yield an average relative improvement of **4.20%** on the MiniF2F-
 483 Test dataset and improve the DeepSeek-Prover-V2’s performance on ProofNet-Test from 22.58% to
 484 **25.81%**. Beyond theorem proving, **GAR** offers a general RL paradigm of co-evolution of the prob-
 485 lem generation and solving under a verifiable environment. It provides a foundation for adversarial
 486 co-training in other reasoning-intensive domains.

486 ETHICS STATEMENT
487488 After carefully reviewing the ethical regulations of the conference, to the best of our knowledge, this
489 work does not present any foreseeable ethical concerns. This research focuses exclusively on formal
490 theorem proving using publicly available mathematical datasets, without involving human subjects,
491 private data, or sensitive content. As far as we are concerned, no negative societal or ethical impacts
492 are anticipated for the contribution of this work. We only use LLMs to polish the writing style and
493 fix grammatical errors in the paper.
494495 REPRODUCIBILITY STATEMENT
496497 We have made efforts to ensure that our work is reproducible. The detailed description of the **GAR**
498 framework, including pseudocode, data source, reward definitions, and training objectives, is pre-
499 sented in Section 2. The experimental setup, including benchmarks, baselines, base model choice,
500 and hyperparameters, is presented in Section 3.1 and Section 3.2. We also plan to further ensure the
501 reproducibility by open-sourcing the code in the near future.
502503 REFERENCES
504505 Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W Ayers, Dragomir Radev, and
506 Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level mathemat-
507 ics. *arXiv preprint arXiv:2302.12433*, 2023.508 Projet Coq. The coq proof assistant-reference manual. *INRIA Rocquencourt and ENS Lyon, version,*
509 5, 1996.
510511 Leonardo De Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The
512 lean theorem prover (system description). In *Automated Deduction-CADE-25: 25th International*
513 *Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings 25*, pp.
514 378–388. Springer, 2015.515 Kefan Dong and Tengyu Ma. Beyond limited data: Self-play llm theorem provers with iterative
516 conjecturing and proving. *arXiv preprint arXiv:2502.00212*, 2025a.
517518 Kefan Dong and Tengyu Ma. Stp: Self-play llm theorem provers with iterative conjecturing and
519 proving. *arXiv preprint arXiv:2502.00212*, 2025b.
520521 Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu,
522 Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms
523 via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.524 John Harrison. Hol light: An overview. In *International Conference on Theorem Proving in Higher*
525 *Order Logics*, pp. 60–66. Springer, 2009.
526527 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,
528 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv*
529 *preprint arXiv:2103.03874*, 2021.530 Albert Q Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée
531 Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem
532 provers with informal proofs. *arXiv preprint arXiv:2210.12283*, 2022.
533534 Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang,
535 Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann
536 Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. [<https://huggingface.co/AI-MO/NuminaMath-CoT>] (https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf), 2024.
537538 Haohan Lin, Zhiqing Sun, Yiming Yang, and Sean Welleck. Lean-star: Learning to interleave
539 thinking and proving. *arXiv preprint arXiv:2407.10040*, 2024.
540

540 Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou
 541 Xia, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover: A frontier model for open-source
 542 automated theorem proving, 2025a. URL <https://arxiv.org/abs/2502.07640>.

543

544 Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan
 545 Geng, Jiawei Ge, Jingruo Sun, et al. Goedel-prover-v2: Scaling formal theorem proving with
 546 scaffolded data synthesis and self-correction. *arXiv preprint arXiv:2508.03613*, 2025b.

547 Leonardo de Moura and Sebastian Ullrich. The lean 4 theorem prover and programming language.
 548 In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, pp. 625–635. Springer, 2021.

549

550 Allen Newell and Herbert Simon. The logic theory machine—a complex information processing
 551 system. *IRE Transactions on information theory*, 2(3):61–79, 1956.

552

553 OpenAI. Learning to reason with llms. <https://openai.com/index/learning-to-reason-with-llms/>, September 13 2024. Accessed: 2024-11-24.

554

555 Lawrence C Paulson. *Isabelle: A generic theorem prover*. Springer, 1994.

556

557 Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving.
 558 *arXiv preprint arXiv:2009.03393*, 2020.

559

560 Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya
 561 Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*,
 562 2022.

563

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

567

568 Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang,
 569 Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical
 570 reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

571

572 Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen,
 573 Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

574

575 Qwen Team. Qwen3, April 2025. URL <https://qwenlm.github.io/blog/qwen3/>.

576

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

580

581 Ruida Wang, Jipeng Zhang, Yizhen Jia, Rui Pan, Shizhe Diao, Renjie Pi, and Tong Zhang. Theorem-
 582 llama: Transforming general-purpose llms into lean4 experts. *arXiv preprint arXiv:2407.03203*,
 583 2024.

584

585 Ruida Wang, Yuxin Li, Yi R Fung, and Tong Zhang. Let's reason formally: Natural-formal hybrid
 586 reasoning enhances llm's math capability. *arXiv preprint arXiv:2505.23703*, 2025b.

587

588 Ruida Wang, Rui Pan, Yuxin Li, Jipeng Zhang, Yizhen Jia, Shizhe Diao, Renjie Pi, Junjie Hu, and
 589 Tong Zhang. Ma-lot: Model-collaboration lean-based long chain-of-thought reasoning enhances
 590 formal theorem proving. *arXiv preprint arXiv:2503.03205*, 2025c.

591

592 Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering
 593 code generation with oss-instruct. *arXiv preprint arXiv:2312.02120*, 2023.

594

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

594 Zijian Wu, Jiayu Wang, Dahua Lin, and Kai Chen. Lean-github: Compiling github lean repositories
 595 for a versatile lean prover. *arXiv preprint arXiv:2407.17227*, 2024b.
 596

597 Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li,
 598 and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale
 599 synthetic data. *arXiv preprint arXiv:2405.14333*, 2024a.

600 Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue
 601 Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback
 602 for reinforcement learning and monte-carlo tree search. *arXiv preprint arXiv:2408.08152*, 2024b.
 603

604 Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and
 605 Kai Shen. Bfs-prover: Scalable best-first tree search for llm-based automatic theorem proving.
 606 *arXiv preprint arXiv:2502.03438*, 2025.

607 Wei Xiong, Jiarui Yao, Yuhui Xu, Bo Pang, Lei Wang, Doyen Sahoo, Junnan Li, Nan Jiang, Tong
 608 Zhang, Caiming Xiong, et al. A minimalist approach to llm reasoning: from rejection sampling
 609 to reinforce. *arXiv preprint arXiv:2504.11343*, 2025.

610 An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu,
 611 Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu,
 612 Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical ex-
 613 pert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024a.
 614

615 Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn
 616 Song. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075*,
 617 2024b.

618 Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil,
 619 Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-
 620 augmented language models. *Advances in Neural Information Processing Systems*, 36, 2024c.
 621

622 Huaiyuan Ying, Zijian Wu, Yihan Geng, Jiayu Wang, Dahua Lin, and Kai Chen. Lean workbook:
 623 A large-scale lean problem set formalized from natural language math problems. *arXiv preprint
 624 arXiv:2406.03847*, 2024.

625 Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. Minif2f: a cross-system benchmark for
 626 formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647

648 A DETAILS OF THE METRICS 649

650 For completeness, this section provides the metric definitions that were omitted due to space limita-
651 tions in Section 3.
652

653 A.1 PASS RATE 654

655 Pass@ x is a widely used metric for evaluating formal theorem provers (Polu et al., 2022; Jiang et al.,
656 2022; Wang et al., 2024; Xin et al., 2024a; Dong & Ma, 2025b; Wang et al., 2025b; Lin et al., 2025b;
657 Wu et al., 2024a; Wang et al., 2025a). For each theorem statement, the model generates x candidate
658 proofs, which are further checked by the Lean4 verifier. If at least one of the x candidates is correct,
659 the theorem is counted as a proved theorem. The pass@ x of the LLM is the fraction of theorems in
660 the dataset with at least one correct proof generated by the prover.
661

662 A.2 AVERAGE PROOF CORRECTNESS RATE 663

664 This metric measures the relative difficulty of a set of statements for a given prover, which is used
665 in Section 3.4). Let \mathcal{D} be a dataset of size m . For each theorem $t_i \in \mathcal{D}$, we sample 16 proofs from
666 the prover and record the number of proofs that pass the verification, which is p_i . The average proof
667 correctness rate is calculated by
668

$$\frac{\sum_{i=1}^m p_i}{16 \cdot m}$$

669 A high value indicates the dataset is easier for the given prover.
670

671 Note that in this experiment, to demonstrate the significance of **GAR** better, we train two extra
672 rounds of Goedel-Prover-V2-8B model.
673

674 A.3 STATEMENT MODIFICATION RATE 675

676 Since current advanced provers have Long Chain-of-thought (CoT) capability, which enables self-
677 reflection, backtracking, and self-correction. In the Long CoT process, a prover may alter the origi-
678 nal statement into a simplified variant in its reasoning trace and ultimately produce a proof for the
679 modified version. To quantify this behavior, we define statement modification rate as the portion
680 of theorems in a dataset that have at least one statement modification across 16 generated proofs.
681 This metric helps assess how often the prover attempts to modify the problem and whether such a
682 behavior may be too significant to cause severe reward hacking.
683

684 B DIRECT COMPARISON WITH RELATED WORKS 685

686 This section aims to provide a more direct comparison between **GAR** and other works that also
687 try to build a dynamic dataset during training, namely Goedel-Prover-V2 (Lin et al., 2025b) and
688 STP (Dong & Ma, 2025b).
689

690 **Compared to Goedel-Prover-V2:** The statement generation in Goedel-V2 relies on a frozen large
691 model to synthesize data for SFT. Crucially, during the RL phase, their statement set remains static.
692 This lack of dynamic updates of statements based on the prover’s evolving capabilities may lead
693 to suboptimal performance and efficiency as the prover outpaces the fixed problem set. In contrast,
694 GAR continuously updates the statement fuser via adversarial RL, ensuring the generated statements
695 remain progressively challenging and aligned with the prover’s current skill level during the RL
696 phase, leading to better empirical performance.
697

698 **Compared to STP:** While STP trains a conjecture model based on the prover’s feedback, it relies
699 on the expert iteration framework based on offline SFT. This process is inherently inefficient because
700 it requires generating a massive volume of training data in a single iteration to achieve effective
701 SFT training. According to Dong & Ma (2025b), they generate 75,000 conjectures per iteration,
702 which is larger than the entire statement set for our generation. Conversely, GAR operates within an
703 online RL cycle. This allows the fuser and prover to continuously update more efficiently, achieving
704

702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
Table 5: pass@32 results on PutnamBench

Model Type	PutnamBench
DeepSeek-Prover-V2-7B (Ren et al., 2025)	22/660
GAR DeepSeek-Prover	24/660

Table 6: Statement Modification Rate for MiniF2F-Test under pass@32

Model Type	Base Model	GAR trained
DeepSeek-Prover-V2 (Ren et al., 2025)	6.96%	13.11%
Goedel-Prover-V2 (Lin et al., 2025b)	24.18%	27.05%

superior performance without the computational burden of the large-scale per-iteration data required by STP.

C ADDITIONAL EXPERIMENTS

This appendix section provides additional experiments that are omitted in the main paper due to space limitations.

C.1 RESULTS ON PUTNAMBENCH

Given that Lean4 formal reasoning is a rapidly progressing field, the MiniF2F-Test serves as a benchmark with high accuracy in current models. To further validate that the **GAR** training can also make the model perform better on more advanced benchmarks, we evaluate both DeepSeek-Prover-V2-7B and GAR-trained DeepSeek-Prover on PutnamBench under pass@32. The results are demonstrated in Table 5.

We can see that the GAR-trained model solves four additional problems compared to the base model on this challenging benchmark. This consistent improvement across MiniF2F, PutnamBench, and ProofNet demonstrates the robustness of the implicit curriculum established by **GAR**.

C.2 EFFICIENCY STUDY

We analyze the computational efficiency of the **GAR** framework, as training from scratch demands substantial resources, to evaluate this, we compare our approach with Kimina-Prover (Wang et al., 2025a), selected for its transparent reporting of training details. The number of roll-out theorems and proofs per iteration in Wang et al. (2025a) matches our configuration. From Figure 4 in their work, we observe that Kimina-Prover achieves approximately 2% of performance gains after 25 training iterations, where **GAR** trained Goedel-Prover-V2-8B reaches more improvements in merely three iterations.

Notably, despite Kimina-Prover starting from an SFT base model, while **GAR** starts from a heavily RL-trained base model, this comparison strengthens our efficiency claim. Because it is well established in prior work (Guo et al., 2025) that models already heavily optimized via RL face diminishing returns and are inherently harder to improve than SFT baselines. The fact that **GAR** achieves relatively significant gains on top of RL-optimized base models in only 3-5 iterations, while Kimina-Prover gains approximately 2% from an SFT starting point over 25 iterations, demonstrates the high sample efficiency of our adversarial training paradigm. Given resource constraints that prevent exhaustive experimentation, we leave the discussion of the scalability of **GAR** for future work.

C.3 STATEMENT MODIFICATION RATE STUDY

To dive deeper into the behavior of statement modification, we provide detailed statement modification rates of both the base model and the **GAR** trained model in MiniF2F-Test under pass@32. The results are shown in Table 6

From the comparison of base models, we can see that the Goedel’s increased statement modification rate happens together with its performance enhancement. We attribute this to the stronger

756 self-correction capability it obtains. Similarly, the soft penalty in GAR ensures that the model is
 757 penalized if it simplifies the problem to a triviality. Furthermore, we can observe that when the
 758 modification rate is low, the GAR training introduces a higher modification rate, as is the case for
 759 DeepSeek-Prover. However, if the modification rate is high in the base model, the soft penalty
 760 will effectively control it within a reasonable range. These findings prove the improvement of the
 761 **GAR**-trained model from another point of view

762

763 C.4 ADDITIONAL ABLATION STUDIES

764

765 C.4.1 FROZEN FUSER STUDY

766 To further analyze our generative adversarial training, we conduct this experiment that applies the
 767 GAR training only on the prover and keeps the statement fuser untrained. We run three iterations
 768 of Frozen fuser GAR training on Goedel-Prover-V2-8B for three iterations. The MiniF2F-Test
 769 pass@32 results are demonstrated in Table 7.

770 From the results, we can see that frozen fuser fails to obtain any performance improvements com-
 771 pared to the base model. This confirms that a static generator is unable to extend the prover’s
 772 capabilities beyond its initial limits. This proves the necessity of co-evolution for both the problem
 773 composer and the prover.

774

775 C.4.2 SINGLE PROBLEM ENHANCEMENT FUISER

776 In order to further study the effectiveness of our idea of problem fusing, which enhances the problem
 777 difficulty by combining two problems into a single one. We conduct the experiment of replacing the
 778 statement fuser with MagicCoder (Wei et al., 2023) that enhances the problem difficulty by restating
 779 the provided single problem to make it more advanced. The results are presented in Table 7.

780 The result indicates that replacing the fusion process with the MagicCoder style problem-evolving
 781 strategy results in a 2.46% performance drop compared to the base model. We conclude this degra-
 782 dation to the limited capability of Qwen3-8B to create more advanced problems with only one
 783 reference. It may not provide sufficient difficulty gradient for the prover, leading to training failure.
 784 In contrast, our fusion approach creates a more effective implicit curriculum by combining distinct
 785 concepts from multiple problems, thereby validating the design of the Statement Fuser.

786

787 C.5 CASE STUDY

788

789 C.5.1 CASE 1:

790 In this section, we demonstrate an example
 791 of how the **GAR** trained model successfully
 792 solves the problem while the base models fail.
 793 Compared to the base prover, the **GAR** trained
 794 model relies less on brute simplification and
 795 more on early structural framing, producing
 796 shorter, steadier proof traces that remain faith-
 797 ful to the statement and are more likely to pass
 798 the verification. If we dig deeper into the Long CoT of two models, we can find that **GAR**-trained
 799 prover writes a shorter and more focused long CoT. While the base model’s CoT is longer but less
 800 directed, it cycles among general identities and simplifications without stabilizing on a closing in-
 801 variant. The result that **GAR**’s trace is both and clearer and more verifiable, which leads to the
 802 successful proof.

803

804 **GAR**-trained model generated result:

805 <|begin_of_sentence|><|User|>Complete the following Lean 4 code:
 806
 807 ````lean4`
 808 `import Mathlib`
 809 `import Aesop`

Table 7: Additional Ablation Studies

Method	MiniF2F-Test
Base model	77.87%
Frozen Fuser	77.87%
MagicCoder Fuser	75.41%
GAR trained	80.33%

```

810 set_option maxHeartbeats 0
811
812 open BigOperators Real Nat Topology Rat
813
814
815 -- Consider a prime $p$ of the form $4 t+1$. Show that $a$ is a
816 → primitive root modulo $p$ iff $-a$ is a primitive root modulo $p$. -/
817 theorem exercise_4_4 {p t: N} (hp0 : p.Prime) (hp1 : p = 4*t + 1)
818   (a : ZMod p) :
819   IsPrimitiveRoot a p  $\Leftrightarrow$  IsPrimitiveRoot (-a) p := by
820    $\dots$ 
821
822 Before producing the Lean 4 code to formally prove the given theorem,
823 → provide a detailed proof plan outlining the main proof steps and
824 → strategies.
825 The plan should highlight key ideas, intermediate lemmas, and proof
826 → structures that will guide the construction of the final formal
827 → proof.
828 <|Assistant|>### Detailed Proof
829
830 **Problem Analysis:**
831 We are given a prime `p` of the form `p = 4 * t + 1` for some natural
832 → number `t`, and an element `a` in the finite field `ZMod p` (where `p`
833 → is the prime number). We need to prove that `a` is a primitive root
834 → modulo `p` if and only if `-a` is a primitive root modulo `p`.
835
836 Recall that `IsPrimitiveRoot a p` means that `a` has multiplicative order
837 → `p - 1 = 4 * t` in the multiplicative group `(ZMod p)ˣ`.
838
839 **Key Observations:**
840 1. The multiplicative group `(ZMod p)ˣ` has order `p - 1 = 4 * t`.
841 2. The negation map `x  $\mapsto$  -x` is an automorphism of `(ZMod p)ˣ` because
842 → `p` is an odd prime (`p = 4 * t + 1 > 2`, and thus `p` is odd). This
843 → means that `-x` has the same order as `x`.
844 3. The multiplicative order of `a` and `-a` are the same if and only if `a
845 → = -a`, but this is not the case unless `p` divides `2 * a`, which is
846 → false here because `a` is arbitrary and `p` is prime. However, the
847 → actual condition is that `IsPrimitiveRoot (-a) p` if and only if
848 → `IsPrimitiveRoot (a) p` because the order of `-a` is the same as the
849 → order of `a` when `p` is of the form `4 * t + 1`.
850
851 But actually, we need a better understanding of the multiplicative group
852 → `(ZMod p)ˣ`.
853
854 For any odd prime `p`, we have the following properties:
855 1. The map `x  $\mapsto$  -x` is an isomorphism on `(ZMod p)ˣ` because `p` is odd.
856   - The inverse is `x  $\mapsto$  -x`, so the map is a permutation of the
857     → multiplicative group.
858 2. For any `x` in `(ZMod p)ˣ`, `x ^ 2` is a square (i.e., `x ^ 2` is a
859 → square in `(ZMod p)ˣ`).
860   - To see this, observe that `(a * b) ^ 2 = a ^ 2 * b ^ 2` and `-1` is a
861     → quadratic residue modulo `p`.
862
863 But for our problem, we don't need these general observations. Instead,
864 → we can use the fact that the order of `-a` is the same as the order
865 → of `a`.
866
867 **Why are the orders the same?**
868
869 Since `p` is an odd prime (`p = 4 * t + 1 > 2`), the multiplicative group
870 → `(ZMod p)ˣ` is cyclic of order `p - 1 = 4 * t`. Let `g` be a generator
871 → of `(ZMod p)ˣ`.
872
873 Then:

```

```

864 1. If `a` is a primitive root (IsPrimitiveRoot a p is true), then `a`  

865  → has order  $4 * t = p - 1$ , so  $^{-a}$  must also have order  $4 * t$   

866  → because  $^{-a}$  is a unit in  $\mathbb{Z}_{\text{Mod } p}$  ( $p$  is prime, so  $^{-a} \neq 0$ ).  

867  - However, this is not correct unless  $-1$  has order  $2$  because  $^{-a} =$   

868  →  $(-1) * a$ , and the order of  $^{-a}$  is  $4 * t$ , and the order of  $-1$   

869  → is  $2$  if  $t \neq 0$  (because  $(-1)^2 = 1$ ). Thus, the order of  $^{-a}$   

870  → is  $\text{lcm}(2, \text{order of } a)$ .  

871 Alternatively, a better approach is to use the following fact:  

872 - The map  $x \mapsto -x$  is an automorphism of the group  $(\mathbb{Z}_{\text{Mod } p})^x$  because  

873  →  $p$  is odd ( $(-1)^2 = 1$  and multiplication is invertible).  

874 **Conclusion:**  

875 Since the map  $x \mapsto -x$  is an automorphism of  $(\mathbb{Z}_{\text{Mod } p})^x$  (because  $p$  is  

876  → an odd prime),  $a$  has order  $4 * t$  if and only if  $^{-a}$  has order  $4$   

877  →  $* t$ . This is because the automorphism  $x \mapsto -x$  preserves the order  

878  → of elements.  

879 Thus, IsPrimitiveRoot a p ( $\text{order } a = 4 * t$ ) is equivalent to  

880  → IsPrimitiveRoot (-a) p ( $\text{order } (-a) = 4 * t$ ).  

881 Therefore, the result follows.  

882 ### Step-by-Step Abstract Plan  

883  

884 1. **Understand the Multiplicative Group**:  

885  - The multiplicative group  $(\mathbb{Z}_{\text{Mod } p})^x$  is cyclic of order  $p - 1 = 4 * t$ .  

886  - Any element has a multiplicative order dividing  $4 * t$ .  

887  

888 2. **Understand the Automorphism  $x \mapsto -x$ **:  

889  - The map  $x \mapsto -x$  is an automorphism of  $(\mathbb{Z}_{\text{Mod } p})^x$  because  $p$  is an  

890  → odd prime ( $(-1)^2 = 1$ ).  

891  

892 3. **Order Preservation**:  

893  - The order of  $^{-a}$  is the same as the order of  $a$ , because  $x \mapsto -x$   

894  → is an automorphism.  

895  

896 4. **Conclusion**:  

897  -  $a$  is a primitive root ( $\text{order } a = p - 1$ ) iff  $^{-a}$  is a primitive  

898  → root ( $\text{order } (-a) = p - 1$ ).  

899 ### Lean 4 `have` Statements  

900  

901 ```lean4  

902 theorem exercise_4_4 {p t:  $\mathbb{N}$ } (hp0 : p.Prime) (hp1 : p = 4*t + 1)  

903  (a :  $\mathbb{Z}_{\text{Mod } p}$ ) :  

904  IsPrimitiveRoot a p  $\leftrightarrow$  IsPrimitiveRoot  $(^{-a}) p$  := by  

905  have h_main : IsPrimitiveRoot a p  $\leftrightarrow$  IsPrimitiveRoot  $(^{-a}) p$  := by  

906  sorry  

907  sorry  

908  ```  

909  

910 ### Complete Lean 4 Proof  

911  

912 ```lean4  

913 theorem exercise_4_4 {p t:  $\mathbb{N}$ } (hp0 : p.Prime) (hp1 : p = 4*t + 1)  

914  (a :  $\mathbb{Z}_{\text{Mod } p}$ ) :  

915  IsPrimitiveRoot a p  $\leftrightarrow$  IsPrimitiveRoot  $(^{-a}) p$  := by  

916  have h_main : IsPrimitiveRoot a p  $\leftrightarrow$  IsPrimitiveRoot  $(^{-a}) p$  := by  

917  haveI : Fact p.Prime := ⟨hp0⟩  

918  have h1 : p  $\neq 0$  := hp0.ne_zero  

919  have h2 : p  $\neq 1$  := hp0.ne_one  

920  have h3 :  $(^{-a} : \mathbb{Z}_{\text{Mod } p}) = -a$  := rfl  

921  constructor

```

```

918
919     . -- Prove the forward direction: if a is a primitive root, then -a
920     -- is a primitive root
921     intro h
922     -- Use the property that the order of -a is the same as the order
923     -- of a
924     have h4 := h.isUnit
925     have h5 := h.pow_eq_one
926     have h6 := h.pow_ne_one_of_pos_of_lt (by decide : (0 : N) < 1) (by
927     rw [← Nat.cast_one]
928     simp_all [ZMod.val_one, Nat.cast_add, Nat.cast_mul,
929     -- Nat.cast_ofNat]
930     <;> omega)
931     simp_all [IsPrimitiveRoot, ZMod.nat_cast_self, Nat.cast_add,
932     -- Nat.cast_mul, Nat.cast_ofNat,
933     ZMod.nat_cast_self]
934     <;>
935     (try omega) <;>
936     (try simp_all [ZMod.nat_cast_self, Nat.cast_add, Nat.cast_mul,
937     -- Nat.cast_ofNat,
938     ZMod.nat_cast_self]) <;>
939     (try
940     {
941         aesop
942     }) <;>
943     (try
944     {
945         simp_all [Units.ext_iff, pow_succ]
946         <;>
947         ring_nf at * <;>
948         omega
949     }) <;>
950     (try
951     {
952         norm_num at *
953         <;>
954         aesop
955     }) <;>
956     (try
957     {
958         simp_all [Units.ext_iff, pow_succ]
959         <;>
960         ring_nf at * <;>
961         omega
962     })
963     <;>
964     aesop
965     . -- Prove the reverse direction: if -a is a primitive root, then a
966     -- is a primitive root
967     intro h
968     -- Use the property that the order of -a is the same as the order
969     -- of a
970     have h4 := h.isUnit
971     have h5 := h.pow_eq_one
972     have h6 := h.pow_ne_one_of_pos_of_lt (by decide : (0 : N) < 1) (by
973     rw [← Nat.cast_one]
974     simp_all [ZMod.val_one, Nat.cast_add, Nat.cast_mul,
975     -- Nat.cast_ofNat]
976     <;> omega)
977     simp_all [IsPrimitiveRoot, ZMod.nat_cast_self, Nat.cast_add,
978     -- Nat.cast_mul, Nat.cast_ofNat,
979     ZMod.nat_cast_self]

```

```

972
973     <;>
974     (try omega) <;>
975     (try simp_all [ZMod.nat_cast_self, Nat.cast_add, Nat.cast_mul,
976     ↪ Nat.cast_ofNat,
977     ↪ ZMod.nat_cast_self]) <;>
978     (try
979     {
980         aesop
981     }) <;>
982     (try
983     {
984         simp_all [Units.ext_iff, pow_succ]
985         <;>
986         ring_nf at * <;>
987         omega
988     }) <;>
989     (try
990     {
991         norm_num at *
992         <;>
993         aesop
994     }) <;>
995     (try
996     {
997         aesop
998     }) <;>
999     (try
1000     {
1001         simp_all [Units.ext_iff, pow_succ]
1002         <;>
1003         ring_nf at * <;>
1004         omega
1005     })
1006     <;>
1007     aesop
1008     exact h_main
1009 ...
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

```

Base model generated result:

```

1004 <|begin_of_sentence|><|User|>Complete the following Lean 4 code:
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025

```

```lean4

```

import Mathlib
import Aesop

set_option maxHeartbeats 0

open BigOperators Real Nat Topology Rat

/- Consider a prime p of the form $4 t+1$. Show that a is a
 primitive root modulo p iff $-a$ is a primitive root modulo p. -/
theorem exercise_4_4 {p t: ℕ} (hp0 : p.Prime) (hp1 : p = 4*t + 1)
 (a : ZMod p) :
 IsPrimitiveRoot a p \leftrightarrow IsPrimitiveRoot (-a) p := by
```


Before producing the Lean 4 code to formally prove the given theorem,  

  → provide a detailed proof plan outlining the main proof steps and  

  → strategies.



The plan should highlight key ideas, intermediate lemmas, and proof  

  → structures that will guide the construction of the final formal  

  → proof.


```

1026 <|Assistant|>### Detailed Proof and Analysis
 1027
 1028 First, let's recall the definitions and properties we will use:
 1029
 1030 1. **Primitive Root**: An element a is a primitive root modulo p if
 \rightarrow the multiplicative order of a in $(\mathbb{Z}/p\mathbb{Z})^*$ is $p - 1$.
 1031 2. **Order of a** : The order of a modulo p , denoted $\text{orderOf } a$, is
 \rightarrow the smallest positive integer k such that $a^k \equiv 1 \pmod{p}$.
 1032 3. **Multiplicative Order**: The multiplicative order of a modulo p is
 \rightarrow the smallest positive integer k such that $a^k \equiv 1 \pmod{p}$.
 1033
 1034 Given p is prime and $p = 4*t + 1$, we need to prove that a is a
 \rightarrow primitive root modulo p if and only if $-a$ is a primitive root
 \rightarrow modulo p .
 1035
 1036 1037
 1038 **Key Observations:**
 1039 1. The multiplicative group $(\mathbb{Z}/p\mathbb{Z})^*$ has order $p - 1 = 4*t$.
 1040 2. The quadratic residues modulo p are $\{a \mid \exists b, b^2 \equiv a \pmod{p}\}$. The
 \rightarrow number of quadratic residues is $(p - 1)/2 = 2*t$ (since $p \equiv 1 \pmod{4}$).
 1041
 1042 3. The multiplicative order of any $a \in (\mathbb{Z}/p\mathbb{Z})^*$ must divide $p - 1 = 4*t$.
 1043
 1044 4. The multiplicative order of $-a$ is the same as the multiplicative
 \rightarrow order of a because $(-a)^2 = a^2 \equiv a^2 \pmod{p}$, and the order is
 \rightarrow determined by the exponentiation.
 1045
 1046
 1047
 1048 **Proof Sketch:**
 1049
 1050 1. Assume a is a primitive root modulo p . This means that a has
 \rightarrow multiplicative order $p - 1 = 4*t$ in $(\mathbb{Z}/p\mathbb{Z})^*$.
 1051 - The order of $-a$ is the same as the order of a , which is $4*t = p - 1$, because:
 1052 - $(-a)^{4*t} = ((-a)^2)^{2*t} = (a^2)^{2*t} \equiv 1 \pmod{p}$ because a^2
 \rightarrow has order $2*t$ (since a is a primitive root and the exponent
 \rightarrow $4*t = 2*(2*t)$, but this is not directly relevant).
 1053 - More precisely, $-a$ and a have the same multiplicative order
 \rightarrow because $(-a)^{4*t} = ((-a)^2)^{2*t} = (a^2)^{2*t} \equiv 1 \pmod{p}$ (if
 \rightarrow a has order $2*t$).
 1054 - However, a simpler approach is to observe that a^2 has order $2*t$
 1055 - and thus $(-a)^2 = a^2$, so $(-a)^{2*t} = (a^2)^t \equiv 1 \pmod{p}$ is not
 1056 - guaranteed. Instead, we can use that $(-a)^{4*t} = ((-a)^2)^{2*t}$
 1057 - $= (a^2)^{2*t} \equiv 1 \pmod{p}$, but this does not directly help unless we
 1058 - know a^2 has order $2*t$.
 1059 - A better approach is to note that $(-a)^k \equiv 1 \pmod{p}$ if and only if
 1060 - $a^k \equiv 1 \pmod{p}$ because:
 1061 - $(-a)^k \equiv 1 \pmod{p}$ is equivalent to $(-1)^k * a^k \equiv 1 \pmod{p}$, so
 1062 - $a^k \equiv (-1)^k \pmod{p}$.
 1063 - Since a is a primitive root, $a^k \equiv 1 \pmod{p}$ if and only if k
 1064 - is a multiple of $4*t$, because a has order $4*t$.
 1065 - Thus, $a^k \equiv (-1)^k \pmod{p}$ would imply $(-1)^k \equiv 1 \pmod{p}$, i.e.,
 1066 - $(-1)^{4*t} \equiv 1 \pmod{p}$, which is true for $p = 4*t + 1$
 1067 - $(-1)^{4*t+1} \equiv -1 \equiv 1 \pmod{p}$? Wait no: $p = 4*t + 1$ and p
 1068 - $\equiv 1 \pmod{4}$, so $(-1)^{4*t+1} = (-1)^{4*t} * (-1) = 1 * (-1) =$
 1069 - $-1 \equiv p - 1 \pmod{p}$, which is not $1 \pmod{p}$. But p is prime and
 1070 - $p \equiv 1 \pmod{4}$, so $-1 \equiv p - 1 \equiv 4*t \equiv 0 \pmod{4}$, which is not
 1071 - helpful. This is a mistake in the reasoning.
 1072 - The correct approach is to note that $(-a)^{4*t} = ((-a)^2)^{2*t} =$
 1073 - $(a^2)^{2*t} \equiv 1 \pmod{p}$ because a^2 has order $2*t$. But a is a
 1074 - primitive root, so $a^{2*t} \equiv 1 \pmod{p}$ ($\text{orderOf } a = 4*t \geq 2*t$).
 1075 - This is not sufficient.
 1076 - Alternatively, observe that:
 1077 - The order of $-a$ divides $4*t$ because the order of a divides
 1078 - $4*t$.
 1079

1080 - The order of $-\alpha$ does not divide $2*t$ unless $\alpha^2 \equiv 1 \pmod{p}$
 1081 \hookrightarrow (since $(-\alpha)^2 = \alpha^2 \equiv 1 \pmod{p}$ would imply $\text{orderOf}(-\alpha) \leq 2$).
 1082 \hookrightarrow But α is a primitive root, so $\alpha^2 \equiv 1 \pmod{p}$ would imply $2 \leq$
 1083 \hookrightarrow $\text{orderOf } \alpha = 4*t$, i.e., $2 \leq 4*t$, which is true since $t \geq 1$.
 1084 \hookrightarrow But $\alpha^2 \equiv 1 \pmod{p}$ is not guaranteed unless $\alpha^2 \equiv 1 \pmod{p}$,
 1085 \hookrightarrow which is not necessarily true.
 1086 - A better way is to use the fact that $-\alpha$ has the same order as α
 1087 \hookrightarrow because $p \equiv 1 \pmod{4}$, and $(-\alpha)^2 = \alpha^2$.
 1088
 1089 However, a simpler and correct approach is to use the following:
 1090 - The multiplicative order of $-\alpha$ modulo p is the same as the
 1091 \hookrightarrow multiplicative order of α modulo p , because $(-\alpha)^2 = \alpha^2$ and
 1092 \hookrightarrow α^2 has order $2*t$ modulo p .
 1093
 1094 But we can use the following:
 1095 - If α is a primitive root modulo p , then for any k , $\alpha^k \equiv 1 \pmod{p}$
 1096 \hookrightarrow if and only if k is a multiple of $4*t$.
 1097 - Then $(-\alpha)^k \equiv 1 \pmod{p}$ is equivalent to $\alpha^k \equiv 1 \pmod{p}$, because
 1098 \hookrightarrow $(-\alpha)^k = (-1)^k * \alpha^k$ and p is odd ($p \equiv 1 \pmod{4}$). So $(-\alpha)^k$
 1099 $\hookrightarrow \equiv 1 \pmod{p}$ is equivalent to $\alpha^k \equiv 1 \pmod{p}$ if p is odd.
 1100 - Hence, $\text{orderOf}(-\alpha) = \text{orderOf } \alpha = 4*t = p - 1$.
 1101
 1102 Therefore, $-\alpha$ is also a primitive root modulo p .
 1103
 1104 #### Correct Approach:
 1105
 1106 1. p is prime and $p \equiv 1 \pmod{4}$, so $(-\alpha)^2 = \alpha^2$ modulo p .
 1107 2. The multiplicative order of $-\alpha$ modulo p is the same as the
 1108 \hookrightarrow multiplicative order of α modulo p , because:
 1109 - $(-\alpha)^k \equiv 1 \pmod{p}$ iff $\alpha^k \equiv 1 \pmod{p}$ for any k , since $(-\alpha)^k =$
 1110 $\hookrightarrow (-1)^k * \alpha^k$, and p is odd ($p \equiv 1 \pmod{4}$).
 1111 - Alternatively, observe that $\text{orderOf}(-\alpha) = \text{orderOf } \alpha$ because $p \equiv$
 1112 $\hookrightarrow 1 \pmod{4}$ and α is a primitive root.
 1113 - In particular, $\text{orderOf}(-\alpha) = p - 1 = \text{orderOf } \alpha$, because $(-\alpha)^{(p-1)} =$
 1114 $\hookrightarrow \alpha^{(p-1)} \equiv 1 \pmod{p}$ (Fermat's Little Theorem).
 1115
 1116 But we need a more precise argument:
 1117 1. If α is a primitive root, then the multiplicative order of α is $p - 1 = 4*t$.
 1118 2. The multiplicative order of $-\alpha$ is the same as the multiplicative
 1119 \hookrightarrow order of α because $p \equiv 1 \pmod{4}$ and α is a primitive root.
 1120 - To see this, note that $(-\alpha)^2 = \alpha^2$, so $\text{orderOf}(-\alpha)$ divides
 1121 $\hookrightarrow \text{orderOf } \alpha^2$ (which is $2*t$).
 1122 - But $\text{orderOf } \alpha^2$ divides $\text{orderOf } \alpha = 4*t$ (since $\alpha^{2t} = (\alpha^2)^{2t} \equiv$
 1123 $\hookrightarrow 1 \pmod{p}$).
 1124 - Also, $\text{orderOf}(-\alpha)$ divides $p - 1 = 4*t$ because $(-\alpha)^{(4*t)} =$
 1125 $\hookrightarrow ((-\alpha)^2)^{(2*t)} = (\alpha^2)^{(2*t)} \equiv 1 \pmod{p}$.
 1126 - Since $\text{orderOf}(-\alpha)$ divides $4*t$ and $4*t$ is the order of α , and
 1127 $\hookrightarrow \text{orderOf}(-\alpha)$ divides $\text{orderOf } \alpha^2$ (which is $2*t$), and orderOf
 1128 $\hookrightarrow \alpha^2$ divides $\text{orderOf } \alpha$ (which is $4*t$), the only common divisors
 1129 \hookrightarrow are 1 and $2*t$.
 1130 - But $\text{orderOf } \alpha = 4*t$ and $\text{orderOf } \alpha^2 = 2*t$.
 1131 - Alternatively, because α is a primitive root, $\alpha^k \equiv 1 \pmod{p}$ iff
 1132 $\hookrightarrow 4*t$ divides k .
 1133 - Similarly, $(-\alpha)^k \equiv 1 \pmod{p}$ iff $4*t$ divides k , because $(-\alpha)^k$
 1134 $\hookrightarrow = \alpha^k$ (since $p \equiv 1 \pmod{4}$).
 1135 - Therefore, the order of $-\alpha$ is the same as the order of α .
 1136
 1137 But we can simplify this:
 1138 - The order of $-\alpha$ is the same as the order of α because $p \equiv 1 \pmod{4}$
 1139 \hookrightarrow and α is a primitive root.
 1140
 1141 Thus, we have:
 1142 1. If α is a primitive root, then $-\alpha$ is also a primitive root.
 1143

```

1134 2. Similarly, if '-a' is a primitive root, then 'a' is also a primitive
1135   ↳ root.
1136
1137 ### Step 1: Abstract Plan
1138
1139 1. **Assume 'a' is a primitive root modulo 'p'**:
1140   - The multiplicative order of 'a' is 'p - 1 = 4*t'.
1141   - We need to show that the multiplicative order of '-a' is '4*t'.
1142   - Use the fact that '(-a)^k ≡ 1 mod p' iff 'a^k ≡ 1 mod p', because 'p
1143   ↳ ≡ 1 mod 4'.
1144   - Thus, the order of '-a' is the same as the order of 'a'.
1145   - Therefore, '-a' is also a primitive root.
1146 2. **Assume '-a' is a primitive root modulo 'p'**:
1147   - Similarly, 'a' is a primitive root.
1148   - The reasoning is symmetric.
1149
1150 ### Step 2: Lean 4 'have' Statements
1151
1152 ```lean4
1153 theorem exercise_4_4 {p t:  $\mathbb{N}$ } (hp0 : p.Prime) (hp1 : p = 4*t + 1)
1154   (a : ZMod p) :
1155     IsPrimitiveRoot a p  $\Leftrightarrow$  IsPrimitiveRoot (-a) p := by
1156     have h_main : IsPrimitiveRoot a p  $\rightarrow$  IsPrimitiveRoot (-a) p := by
1157       sorry
1158     have h_converse : IsPrimitiveRoot (-a) p  $\rightarrow$  IsPrimitiveRoot a p := by
1159       sorry
1160     have h_final : IsPrimitiveRoot a p  $\Leftrightarrow$  IsPrimitiveRoot (-a) p := by
1161       sorry
1162     exact h_final
1163 ```
1164
1165 ### Explanation:
1166 1. 'h_main': If 'a' is a primitive root, then '-a' is also a primitive
1167   ↳ root.
1168 2. 'h_converse': If '-a' is a primitive root, then 'a' is also a primitive
1169   ↳ root.
1170 3. 'h_final': Combine the two directions to get the full biconditional.
1171
1172 ### Complete Lean 4 Proof
1173
1174 ```lean4
1175 theorem exercise_4_4 {p t:  $\mathbb{N}$ } (hp0 : p.Prime) (hp1 : p = 4*t + 1)
1176   (a : ZMod p) :
1177     IsPrimitiveRoot a p  $\Leftrightarrow$  IsPrimitiveRoot (-a) p := by
1178     have h_main : IsPrimitiveRoot a p  $\rightarrow$  IsPrimitiveRoot (-a) p := by
1179       intro h
1180       haveI := Fact.mk hp0
1181       have h1 := h.isUnit
1182       have h2 := h.pow_eq_one
1183       have h3 := h.pow_ne_one_of_pos_of_lt (by decide : (0 :  $\mathbb{N}$ ) < 1) (by
1184         simp_all [hp1, Nat.Prime.one_lt]
1185         <;> omega)
1186       have h4 := h.pow_ne_one_of_pos_of_lt (by decide : (0 :  $\mathbb{N}$ ) < 2) (by
1187         simp_all [hp1, Nat.Prime.one_lt]
1188         <;> omega)
1189       simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1190       <;>
1191       (try omega) <;>
1192       (try simp_all [IsPrimitiveRoot, pow_mul, pow_add, pow_one, pow_two,
1193         ↳ ZMod.nat_cast_self]) <;>
1194       (try omega) <;>
1195       (try
1196         {
1197           have h5 := h.pow_eq_one
1198           have h6 := h.isUnit

```

```

1188         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1189         <;>
1190         aesop
1191     }) <;>
1192     (try
1193     {
1194         have h5 := h.pow_eq_one
1195         have h6 := h.isUnit
1196         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1197         <;>
1198         aesop
1199     })
1200     <;>
1201     (try
1202     {
1203         aesop
1204     })
1205     <;>
1206     (try
1207     {
1208         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1209         <;>
1210         aesop
1211     })
1212     <;>
1213     (try
1214     {
1215         omega
1216     })
1217     <;>
1218     (try
1219     {
1220         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1221         <;>
1222         omega
1223     })
1224
have h_converse : IsPrimitiveRoot (¬a) p → IsPrimitiveRoot a p := by
1225     intro h
1226     haveI := Fact.mk hp0
1227     have h1 := h.isUnit
1228     have h2 := h.pow_eq_one
1229     have h3 := h.pow_ne_one_of_pos_of_lt (by decide : (0 : N) < 1) (by
1230         simp_all [hp1, Nat.Prime.one_lt]
1231         <;> omega)
1232     have h4 := h.pow_ne_one_of_pos_of_lt (by decide : (0 : N) < 2) (by
1233         simp_all [hp1, Nat.Prime.one_lt]
1234         <;> omega)
1235     simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1236     <;>
1237     (try omega) <;>
1238     (try simp_all [IsPrimitiveRoot, pow_mul, pow_add, pow_one, pow_two,
1239     ← ZMod.nat_cast_self]) <;>
1240     (try omega) <;>
1241     (try
1242     {
1243         have h5 := h.pow_eq_one
1244         have h6 := h.isUnit
1245         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1246         <;>

```

```

1242         aesop
1243     }) <;>
1244     (try
1245     {
1246         have h5 := h.pow_eq_one
1247         have h6 := h.isUnit
1248         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1249         <;>
1250         aesop
1251     })
1252     <;>
1253     (try
1254     {
1255         aesop
1256     })
1257     <;>
1258     (try
1259     {
1260         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1261         <;>
1262         aesop
1263     })
1264     <;>
1265     (try
1266     {
1267         omega
1268     })
1269     <;>
1270     (try
1271     {
1272         simp_all [ZMod.nat_cast_self, pow_one, pow_two, neg_neg]
1273         <;>
1274         omega
1275     })
1276     <;>
1277     aesop

1278 have h_final : IsPrimitiveRoot a p  $\leftrightarrow$  IsPrimitiveRoot ( $\neg$ a) p := by
1279   constructor
1280   · exact h_main
1281   · exact h_converse
1282
1283
1284 C.5.2 CASE 2:
1285

```

In another ProofNet example, **GAR** again succeeds where the base prover fails. This instance more clearly illustrates how **GAR** mitigates useless simplifications. It uses NLs to reach a comprehensive proof plan and effectively conduct the proof in a relatively simple manner. However, the base model conducts a quicker search but a prolonged proof. It expends effort on low-yield algebra and yields lengthy, fragile derivations. These observations highlight that the implicit curriculum in **GAR** fosters more efficient reasoning paths and greater verification robustness.

GAR-trained model generated result:

```

1293
1294 <|begin_of_sentence|><|User|>Complete the following Lean 4 code:
1295
1296 ```lean4

```

```

1296 import Mathlib
1297 import Aesop
1298
1299 set_option maxHeartbeats 0
1300
1301 open BigOperators Real Nat Topology Rat
1302
1303
1304 --- Prove that  $x^4+4x^3+6x^2+2x+1$  is irreducible in  $\mathbb{Z}[x]$ . --
1305 theorem exercise_9_4_2c : Irreducible
1306   (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial ℤ) := by
1307   ...
1308
1309 Before producing the Lean 4 code to formally prove the given theorem,
1310   → provide a detailed proof plan outlining the main proof steps and
1311   → strategies.
1312 The plan should highlight key ideas, intermediate lemmas, and proof
1313   → structures that will guide the construction of the final formal
1314   → proof.
1315 <|Assistant|>### Detailed Proof
1316
1317 **Problem:** Prove that  $\langle P(x) = x^4 + 4x^3 + 6x^2 + 2x + 1 \rangle$  is
1318   → irreducible in  $\langle \mathbb{Z}[x] \rangle$ .
1319
1320 **Approach:**
1321 We will use the Rabinowitz Criterion (also known as the "Rabinowitz
1322   → irreducibility test" or "Rabinowitz's Criterion") to prove that  $\langle$ 
1323   →  $P(x) \rangle$  is irreducible over  $\langle \mathbb{Z}[x] \rangle$ .
1324
1325 **Rabinowitz Criterion:**
1326 Let  $\langle P(x) = x^n + a_{n-1}x^{n-1} + \dots + a_0 \rangle$  be a polynomial in
1327   →  $\langle \mathbb{Z}[x] \rangle$ . Suppose that there exists a prime  $\langle p \rangle$  such
1328   → that:
1329 1.  $\langle p \rangle$  divides each  $\langle a_i \rangle$  for  $\langle 0 \leq i \leq n-1 \rangle$ .
1330 2.  $\langle p^2 \rangle$  does not divide  $\langle a_0 \rangle$ .
1331 Then  $\langle P(x) \rangle$  is irreducible over  $\langle \mathbb{Z}[x] \rangle$ .
1332
1333 **Application to  $\langle P(x) \rangle$ :
1334 Let  $\langle P(x) = x^4 + 4x^3 + 6x^2 + 2x + 1 \rangle$ .
1335
1336 1. The constant term is  $\langle a_0 = 1 \rangle$ .
1337 2. The primes dividing all coefficients  $\langle a_0, a_1, a_2, a_3 \rangle$  are the
1338   → primes dividing  $\langle \gcd(1, 2, 6, 4, 1) = 1 \rangle$ . So, no primes satisfy
1339   → the first condition.
1340   → Hmm, this is incorrect: actually, all coefficients  $\langle a_0 = 1 \rangle$ ,  $\langle$ 
1341   →  $a_1 = 2 \rangle$ ,  $\langle a_2 = 6 \rangle$ ,  $\langle a_3 = 4 \rangle$  are divisible by  $\langle 1 \rangle$ ,
1342   → but the second condition is about  $\langle a_0 = 1 \rangle$ , not all other
1343   → coefficients. This suggests that perhaps we need a different
1344   → approach.
1345
1346 But the Rabinowitz Criterion can still be applied:
1347 Consider  $\langle p = 2 \rangle$ .
1348   →  $\langle 2 \rangle$  divides  $\langle a_0 = 1 \rangle$  (False).
1349   →  $\langle 2 \rangle$  divides  $\langle a_1 = 2 \rangle$  (True).
1350   →  $\langle 2 \rangle$  divides  $\langle a_2 = 6 \rangle$  (True).
1351   →  $\langle 2 \rangle$  divides  $\langle a_3 = 4 \rangle$  (True).
1352 But  $\langle p^2 = 4 \rangle$  divides  $\langle a_0 = 1 \rangle$  (False).
1353
1354 This does not work. So, we need another prime.
1355
1356 Consider  $\langle p = 3 \rangle$ :
1357   →  $\langle 3 \rangle$  divides  $\langle a_0 = 1 \rangle$  (No).
1358   →  $\langle 3 \rangle$  divides  $\langle a_1 = 2 \rangle$  (No).
1359   →  $\langle p \rangle$  does not divide any of the coefficients, so this is invalid.

```

```

1350
1351 Consider  $\langle p = 5 \rangle$ :
1352 -  $\langle 5 \rangle$  divides  $\langle a_0 = 1 \rangle$  (No).
1353 -  $\langle 5 \rangle$  divides  $\langle a_1 = 2 \rangle$  (No).
1354  $\langle p \rangle$  does not divide any of the coefficients, so this is invalid.

1355 **Alternative Approach:**
1356 Let's consider the polynomial  $\langle Q(x) = x^4 + 4x^3 + 6x^2 + 2x + 1 \rangle$ . We
1357 → can attempt to factor it or check for possible roots.
1358
1359 But  $\langle Q(0) = 1 \rangle$ ,  $\langle Q(1) = 1 + 4 + 6 + 2 + 1 = 14 \rangle$ , and  $\langle Q(-1) = 1$ 
1360 →  $- 4 + 6 - 2 + 1 = 2 \rangle$ . No simple rational roots.

1361 Alternatively, perhaps we can factor  $\langle Q(x) \rangle$  into quadratics.
1362
1363 Assume  $\langle Q(x) = (x^2 + a x + b)(x^2 + c x + d) \rangle$ .
1364
1365 Expanding gives:
1366  $\langle [x^4 + (a + c)x^3 + (ac + b + d)x^2 + (ad + bc)x + bd = Q(x)] \rangle$ 
1367
1368 Thus, we get the system:
1369 1.  $\langle a + c = 4 \rangle$ 
1370 2.  $\langle ac + b + d = 6 \rangle$ 
1371 3.  $\langle ad + bc = 2 \rangle$ 
1372 4.  $\langle bd = 1 \rangle$ 
1373
1374 From  $\langle bd = 1 \rangle$ , since  $\langle b, d \in \mathbb{Z} \rangle$ , we have the following
1375 → cases:
1376 1.  $\langle b = 1, d = 1 \rangle$ , or
1377 2.  $\langle b = -1, d = -1 \rangle$ .
1378
1379 **Case 1:  $\langle b = d = 1 \rangle$ **
1380 From (1):  $\langle a + c = 4 \rangle$ 
1381 From (2):  $\langle ac + 2 = 6 \rangle$  implies  $ac = 4$ 
1382 From (3):  $\langle a + c = 4 \rangle$ ,  $\langle ac = 4 \rangle$ 
1383 But  $\langle a, c \rangle$  are roots of  $\langle t^2 - 4t + 4 = 0 \rangle$ , i.e.,  $\langle t = 2 \rangle$ . So
1384 →  $\langle a = c = 2 \rangle$ .
1385
1386 This satisfies all equations:
1387 -  $\langle a + c = 4 \rangle$ 
1388 -  $\langle ac = 4 \rangle$ 
1389 -  $\langle ad + bc = 2 \cdot 1 + 2 \cdot 1 = 4 \neq 2 \rangle$  (Does not hold).
1390
1391 Wait, this is incorrect. The problem is in (3):
1392  $\langle ad + bc = a \cdot 1 + c \cdot 1 = (a + c) = 4 \neq 2 \rangle$ , which
1393 → contradicts the third condition.
1394
1395 Hence, this case is invalid.
1396
1397 **Case 2:  $\langle b = d = -1 \rangle$ **
1398 From (1):  $\langle a + c = 4 \rangle$ 
1399 From (2):  $\langle ac - 2 = 6 \rangle$  implies  $ac = 8$ 
1400 From (3):  $\langle -a + -c = 2 \rangle$  implies  $a + c = -2$ , but this contradicts
1401 →  $a + c = 4$ .
1402
1403 This case is also invalid.

1404
1405 **Another Approach: Eisenstein's Criterion**
1406 This polynomial is not directly suited for Eisenstein's Criterion,
1407 → because replacing  $\langle x \rangle$  with  $\langle x + 1 \rangle$  gives:
1408  $\langle [ (x + 1)^4 + 4(x + 1)^3 + 6(x + 1)^2 + 2(x + 1) + 1 \rangle$ 
1409  $\langle [ = x^4 + 4x^3 + 6x^2 + 4x + 1 + 4x^3 + 12x^2 + 12x + 4 + 6x^2 + 12x + 6$ 
1410 →  $+ 2x + 2 + 1 \rangle$ 
1411  $\langle [ = x^4 + (4x^3 + 4x^3) + (6x^2 + 12x^2 + 6x^2) + (4x + 12x + 12x + 2x)$ 
1412 →  $+ (1 + 4 + 6 + 2 + 1) \rangle$ 

```

```

1404 \[ = x^4 + 8x^3 + 24x^2 + 30x + 14 \]
1405 But we can try another substitution.
1406
1407 Alternatively, observe that:
1408 \[ Q(x) = x^4 + 4x^3 + 6x^2 + 2x + 1 \]
1409 \[ = (x^2 + 2x)^2 + 2x^2 + 2x + 1 \]
1410 But this seems not helpful.
1411
1412 However, a better approach is to note that:
1413 \[ Q(x) = (x^2 + 2x + 1)^2 - (2x^2 + 1) \]
1414 But:
1415 \[ Q(x) = x^4 + 4x^3 + 6x^2 + 2x + 1 \]
1416 \[ = (x^2 + 2x + 1)^2 - (2x + 2) \]
1417 But \[ (x^2 + 2x + 1)^2 - (2x + 2) = x^4 + 4x^3 + 6x^2 + 4x + 1 - 2x - 2 \]
1418 \[ \rightarrow = x^4 + 4x^3 + 6x^2 + 2x - 1 \neq Q(x) \].
1419
1420 This substitution is incorrect.
1421
1422 **Conclusion from the failed attempts:**
1423 It seems difficult to factor \[ Q(x) \] in \[ \mathbb{Z}[x] \] by simple
1424 \[ \rightarrow \] methods.
1425
1426 But we can try a more general approach using the Cohn's Irreducibility
1427 \[ \rightarrow \] Criterion or Ritt's Criterion, but these are not straightforward.
1428
1429 However, we can use the Rational Root Test (or simply checking small
1430 \[ \rightarrow \] primes):
1431 Check for \[ x = \pm 1, \pm p \] for primes \[ p \]:
1432 - \[ Q(1) = 1 + 4 + 6 + 2 + 1 = 14 \]
1433 - \[ Q(-1) = 1 - 4 + 6 - 2 + 1 = 2 \]
1434 - \[ Q(2) = 16 + 32 + 24 + 4 + 1 = 77 \]
1435 - \[ Q(-2) = 16 - 32 + 24 - 4 + 1 = 5 \]
1436
1437 Since \[ Q(x) \] has no rational roots, it is irreducible over
1438 \[ \rightarrow \] \[ \mathbb{Z} \].
1439
1440 But this seems incorrect, as \[ Q(x) \] is reducible for \[ x = 0 \] and
1441 \[ \rightarrow \] \[ x = -1 \], but we are not using this approach.
1442
1443 Alternatively, we can use the **Schinzel's criterion** or **Rabinowitz
1444 \[ \rightarrow \] Criterion** again:
1445 Consider \[ Q(x + 1) = (x + 1)^4 + 4(x + 1)^3 + 6(x + 1)^2 + 2(x + 1) + 1 \]
1446 \[ \rightarrow \]
1447 \[ = x^4 + 4x^3 + 6x^2 + 4x + 1 + 4x^3 + 12x^2 + 12x + 4 + 6x^2 + 12x + 6 \]
1448 \[ \rightarrow + 2x + 2 + 1 \]
1449 \[ = x^4 + (4x^3 + 4x^3) + (6x^2 + 12x^2 + 6x^2) + (4x + 12x + 12x + 2x) \]
1450 \[ \rightarrow + (1 + 4 + 6 + 2 + 1) \]
1451 \[ = x^4 + 8x^3 + 24x^2 + 30x + 14 \]
1452
1453 Hmm, no immediate factorization here either.
1454
1455 **Final Attempt:**
1456 We use the fact that if \[ Q(x) \] is irreducible over \[ \mathbb{Z} \],
1457 \[ \rightarrow \] then it is irreducible over \[ \mathbb{Q} \].
1458
1459 Alternatively, we can directly check that \[ Q(x) \] has no factors in
1460 \[ \rightarrow \] \[ \mathbb{Z}[x] \] other than \[ Q(x) \] itself and constants.
1461
1462 We can perform a complete check:
1463
1464 Suppose \[ Q(x) = A(x) B(x) \] is a factorization into non-constant
1465 \[ \rightarrow \] polynomials.

```

1458 We can write:
 1459 $\backslash [A(x) = x^4 + a_1 x^3 + a_2 x^2 + a_3 x + a_4 \backslash]$
 1460 $\backslash [B(x) = 1 \backslash]$
 1461 But $\backslash (B(x) \backslash)$ is a unit and $\backslash (\deg(B(x)) = 0 \backslash)$, so this is trivial.
 1462 Alternatively, assume $\backslash (A(x) \backslash)$ has degree $\backslash (1 \backslash)$ or $\backslash (2 \backslash)$, etc.
 1463 But this is tedious and unlikely to work.
 1464 Given that the problem is from an older source, we can infer that the
 1465 \hookrightarrow polynomial $\backslash (Q(x) \backslash)$ is irreducible.
 1466 But in Lean, we can proceed by proving that $\backslash (Q(x) \backslash)$ has no non-trivial
 1467 \hookrightarrow factors, by checking that for every pair of integers $\backslash ((a, b) \backslash)$
 1470 \hookrightarrow such that $\backslash (b \neq \pm 1 \backslash)$, we cannot write $\backslash (Q(x) \backslash)$ as a product
 1471 \hookrightarrow of two non-constant polynomials.
 1472 Alternatively, we can use Lean's libraries to prove irreducibility by
 1473 \hookrightarrow contradiction:
 1474
****Proof Sketch:****
 1475 1. Assume $\backslash (Q(x) = A(x) B(x) \backslash)$.
 1476 2. Look at the constant term.
 1477 The constant term is $\backslash (1 \backslash)$, so possible constant factors are $\backslash (\pm 1 \backslash)$.
 1478 3. So either $\backslash (A(x) \backslash)$ or $\backslash (B(x) \backslash)$ is a constant, say $\backslash (A(x) = \pm 1 \backslash)$.
 1479 4. But then $\backslash (Q(x) \backslash)$ would have to be constant, which is false.
 1480 So, contradiction.
 1481
 1482 But Lean may not directly have lemmas for this.
 1483
 1484 A better approach is to use that for a monic polynomial with integer
 1485 \hookrightarrow coefficients and $\backslash (\pm 1 \backslash)$ constant term, it is irreducible if it
 1486 \hookrightarrow has no non-trivial factors.
 1487
 1488 But in Lean, we can prove that no non-trivial factors exist.
 1489
 1490 However, checking all possible factors is impractical.
 1491
 1492 Alternatively, we can use the fact that $\backslash (Q(x) \backslash)$ is a cyclotomic-like
 1493 \hookrightarrow polynomial or a Schur polynomial.
 1494 But since this seems too involved, we can use Lean's ****irreducibility**
 1495 \hookrightarrow **testing** (but that's not part of standard libraries).
 1496
****Alternative Step:****
 1497 We can instead show that $\backslash (Q(x) \backslash)$ is not a perfect square in $\backslash ($
 1498 \hookrightarrow $\mathbb{Z}[x]$ $\backslash)$.
 1499
 1500 But this is non-trivial.
 1501
 1502 Alternatively, we can use ****Mignotte's bound**** to check that no
 1503 \hookrightarrow non-trivial factors exist.
 1504 But given that $\backslash (Q(x) \backslash)$ is not a known factorizable polynomial, we can
 1505 \hookrightarrow simply use the fact that it is not reducible and is of degree $\backslash (4 \backslash)$,
 1506 \hookrightarrow so it must be irreducible if it has no roots in $\backslash (\mathbb{Z} \backslash)$.
 1507 Since $\backslash (Q(x) \backslash)$ is strictly increasing for $\backslash (x \geq 0 \backslash)$ and $\backslash (Q(-1) = 2 \backslash)$, $\backslash (Q(0) = 1 \backslash)$, and $\backslash (Q(1) = 14 \backslash)$, there are no integer
 1509 \hookrightarrow roots.
 1510
 1511 But Lean can confirm this by checking values.

```

1512 But instead, we can use:
1513
1514 **Lemmas:**
1515 1. If  $\langle Q(x) \rangle$  is reducible in  $\langle \mathbb{Z}[x] \rangle$ , then it has a
1516    ↳ linear or quadratic factor.
1517 2. A reducible polynomial must have a factor with  $\langle \deg \leq \lfloor$ 
1518    ↳  $\frac{\deg(Q)}{2} \rfloor = 2 \rangle$ .
1519 3. The possible quadratics are  $\langle x^2 + ax + 1 \rangle$  or similar, but none
1520    ↳ divide  $\langle Q(x) \rangle$ .
1521
1522 But checking this is tedious.
1523
1524 Instead, we can appeal to a known fact that  $\langle x^4 + x^3 + x^2 + x + 1 \rangle$ 
1525    ↳ is irreducible, and substituting  $\langle x + 1 \rangle$  gives:
1526  $\langle (x + 1)^4 + 4(x + 1)^3 + 6(x + 1)^2 + 2(x + 1) + 1 \rangle$ 
1527  $\langle = x^4 + 4x^3 + 6x^2 + 4x + 1 + 4x^3 + 12x^2 + 12x + 4 + 6x^2 + 12x + 6$ 
1528    ↳ + 2x + 2 + 1 \rangle
1529  $\langle = x^4 + 8x^3 + 24x^2 + 30x + 14 \rangle$ 
1530
1531 This is the same as above.
1532
1533 Alternatively, check that  $\langle x^4 + 4x^3 + 6x^2 + 2x + 1 \rangle$  is irreducible
1534    ↳ using:
1535
1536 **Useful Lemma:** The polynomial  $\langle x^4 + ax^3 + bx^2 + cx + 1 \rangle$  is
1537    ↳ irreducible in  $\langle \mathbb{Z}[x] \rangle$  if  $\langle a^2 < 4b \rangle$  or  $\langle b^2 < 4c \rangle$ 
1538    ↳ , etc.
1539
1540 But this is not directly helpful.
1541
1542 **Lean-Applicable Approach:**
1543 To prove irreducibility in Lean, we can directly use:
1544 - The reduction modulo primes approach to eliminate possible
1545    ↳ factorizations.
1546
1547 Alternatively, we can use the fact that  $\langle Q(x) \rangle$  is a cyclotomic-like
1548    ↳ polynomial or a Cohn polynomial.
1549
1550 But given that the polynomial is of degree  $\langle 4 \rangle$  and no obvious
1551    ↳ factorization exists, we can proceed by contradiction.
1552
1553 ### Step-by-Step Abstract Plan
1554
1555 1. **Assume Reducibility:** Suppose  $\langle Q(x) \rangle$  is reducible in  $\langle$ 
1556    ↳  $\mathbb{Z}[x] \rangle$ . Then,  $\langle Q(x) = A(x) B(x) \rangle$  with  $\langle \deg(A(x))$ 
1557    ↳  $\geq 1 \rangle$ ,  $\langle \deg(B(x)) \geq 1 \rangle$ , and  $\langle \deg(A(x)) + \deg(B(x)) =$ 
1558    ↳ 4 \rangle.
1559
1560 2. **Analyze Constant Terms:** Since  $\langle Q(0) = 1 \rangle$  and  $\langle Q(x) \rangle$  has no
1561    ↳ rational roots, no constant factors can exist unless one of  $\langle A(x)$ 
1562    ↳ \rangle or  $\langle B(x) \rangle$  is  $\langle \pm 1 \rangle$ .
1563
1564 3. **Check for Linear Factors:** If  $\langle \deg(A(x)) = 1 \rangle$ , say  $\langle A(x) = x$ 
1565    ↳ + a \rangle, then  $\langle Q(x) = (x + a) B(x) \rangle$ . Expand to get coefficients
1566    ↳ and contradict.
1567
1568 4. **Check for Quadratic Factors:** If  $\langle \deg(A(x)) = 2 \rangle$ , say  $\langle A(x)$ 
1569    ↳ = x^2 + ax + b \rangle, analyze to find contradictions.
1570
1571 5. **Contradiction:** Since all possible factorizations lead to
1572    ↳ contradictions,  $\langle Q(x) \rangle$  must be irreducible.
1573
1574 ### Lean 4 Abstract Plan with `have` Statements
1575
1576 ```lean4
1577 theorem exercise_9_4_2c : Irreducible

```

```

1566   (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial ℤ) := by
1567   have h_main : Irreducible (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial
1568   ← ℤ) := by
1569   sorry
1570   sorry
1571   ...
1572 ### Complete Lean 4 Proof
1573
1574 ```lean4
1575 theorem exercise_9_4_2c : Irreducible
1576   (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial ℤ) := by
1577   have h_main : Irreducible (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial
1578   ← ℤ) := by
1579   apply?
1580   apply h_main
1581   ...

```

Base model generated result:

```

1582 <|begin_of_sentence|><|User|>Complete the following Lean 4 code:
1583
1584 ```lean4
1585 import Mathlib
1586 import Aesop
1587
1588 set_option maxHeartbeats 0
1589
1590 open BigOperators Real Nat Topology Rat
1591
1592
1593
1594 -- Prove that $x^4+4x^3+6x^2+2x+1$ is irreducible in $\mathbb{Z}[x]$. --
1595 theorem exercise_9_4_2c : Irreducible
1596   (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial ℤ) := by
1597   ...
1598
1599 Before producing the Lean 4 code to formally prove the given theorem,
1600   → provide a detailed proof plan outlining the main proof steps and
1601   → strategies.
1602 The plan should highlight key ideas, intermediate lemmas, and proof
1603   → structures that will guide the construction of the final formal
1604   → proof.
1605 <|Assistant|>### Detailed Proof and Analysis
1606
1607 We need to prove that the polynomial  $\langle P(x) = x^4 + 4x^3 + 6x^2 + 2x + 1 \rangle$ 
1608   → is irreducible over  $\langle \mathbb{Z}[x] \rangle$ .
1609
1610 #### Step 1: Understand the Polynomial
1611 First, observe that  $\langle P(x) \rangle$  is palindromic. A palindromic polynomial
1612   →  $\langle P(x) \rangle$  can be factored as  $\langle P(x) = (x^2 + a x + b)(x^2 + c x +$ 
1613   →  $d) \rangle$  if and only if  $\langle P(x) \rangle$  can be written as a product of two
1614   → quadratic polynomials.
1615
1616 #### Step 2: Try to Factor  $\langle P(x) \rangle$ 
1617 Assume that  $\langle P(x) \rangle$  factors into two non-constant polynomials with
1618   → integer coefficients:
1619  $\langle P(x) = (x^2 + a x + b)(x^2 + c x + d). \rangle$ 
1620
1621 Expanding the right-hand side gives:
1622  $\langle x^4 + (a + c)x^3 + (ac + b + d)x^2 + (ad + bc)x + bd. \rangle$ 
1623
1624 By comparing coefficients, we get the system:
1625 1.  $\langle a + c = 4 \rangle$ ,

```

```

1620 2.  $\backslash ( ac + b + d = 6 \backslash ),$ 
1621 3.  $\backslash ( ad + bc = 2 \backslash ),$ 
1622 4.  $\backslash ( bd = 1 \backslash ).$ 
1623
1624 #### Step 3: Solve the System
1625 From  $\backslash ( bd = 1 \backslash )$ , the possible integer pairs  $\backslash ((b, d) \backslash)$  are:
1626  $\backslash [ (1, 1), (-1, -1) \backslash ]$ 
1627
1628 **Case 1:**  $\backslash ( b = 1 \backslash ), \backslash ( d = 1 \backslash ).$ 
1629 Substitute into the other equations:
1630 1.  $\backslash ( a + c = 4 \backslash ),$ 
1631 2.  $\backslash ( ac + 1 + 1 = 6 \Rightarrow ac = 4 \backslash ),$ 
1632 3.  $\backslash ( a + c = 4 \backslash ),$ 
1633 4.  $\backslash ( ad + bc = a + c = 4 \backslash ),$  but this is already satisfied.
1634
1635 But from  $\backslash ( a + c = 4 \backslash )$  and  $\backslash ( ac = 4 \backslash ),$  the quadratic equation for  $\backslash ($ 
1636  $\hookrightarrow a \backslash )$  is:
1637  $\backslash [ t^2 - 4t + 4 = 0 \Rightarrow (t - 2)^2 = 0 \Rightarrow t = 2. \backslash ]$ 
1638 Thus  $\backslash ( a = c = 2 \backslash ).$ 
1639
1640 Check the last equation:  $\backslash ( ad + bc = 2 \cdot 1 + 2 \cdot 1 = 4 \neq 2 \backslash )$ 
1641  $\hookrightarrow \backslash ).$  This is a contradiction, so this case is invalid.
1642
1643 **Case 2:**  $\backslash ( b = -1 \backslash ), \backslash ( d = -1 \backslash ).$ 
1644 Substitute into the other equations:
1645 1.  $\backslash ( a + c = 4 \backslash ),$ 
1646 2.  $\backslash ( ac - 1 - 1 = 6 \Rightarrow ac = 8 \backslash ),$ 
1647 3.  $\backslash ( ad + bc = -a - c = 2 \backslash ),$ 
1648 4.  $\backslash ( bd = 1 \backslash ).$ 
1649
1650 But from  $\backslash ( a + c = 4 \backslash )$  and  $\backslash ( ac = 8 \backslash ),$  the quadratic equation for  $\backslash ($ 
1651  $\hookrightarrow a \backslash )$  is:
1652  $\backslash [ t^2 - 4t + 8 = 0, \backslash ]$ 
1653 which has discriminant  $\backslash ( D = 16 - 32 = -16 < 0 \backslash ).$  Thus, there are no
1654  $\hookrightarrow$  real (and hence no integer) roots, so this case is invalid.
1655
1656 #### Step 4: Check Other Possible Factorizations
1657 The above exhaustive search shows that  $\backslash ( P(x) \backslash )$  does not factor into
1658  $\hookrightarrow$  two non-constant polynomials with integer coefficients. Thus,  $\backslash ( P(x) \backslash )$ 
1659  $\hookrightarrow \backslash )$  is irreducible.
1660
1661 #### Verification of the Irreducibility
1662 We have checked all possible factorizations of  $\backslash ( P(x) \backslash )$  into quadratics
1663  $\hookrightarrow$  and found that none satisfy the coefficient conditions. Therefore,  $\backslash ($ 
1664  $\hookrightarrow P(x) \backslash )$  is irreducible.
1665
1666 ### Abstract Plan
1667
1668 1. **Assume Factorization**: Suppose  $\backslash ( P(x) = (x^2 + a x + b)(x^2 + c x$ 
1669  $\hookrightarrow + d) \backslash )$  for integers  $\backslash ( a, b, c, d \backslash ).$ 
1670 2. **Compare Coefficients**: Derive the system of equations for  $\backslash ( a, b,$ 
1671  $\hookrightarrow c, d \backslash ).$ 
1672  $\quad \backslash ( a + c = 4 \backslash ),$ 
1673  $\quad \backslash ( ac + b + d = 6 \backslash ),$ 
1674  $\quad \backslash ( ad + bc = 2 \backslash ),$ 
1675  $\quad \backslash ( bd = 1 \backslash ).$ 
1676 3. **Solve  $\backslash ( bd = 1 \backslash )**$ : Consider the pairs  $\backslash ((b, d) \backslash)$  as  $\backslash ((1, 1) \backslash )$ 
1677  $\hookrightarrow$  and  $\backslash ((-1, -1) \backslash ).$ 
1678 4. **Case  $\backslash ( b = 1 \backslash ), \backslash ( d = 1 \backslash )**$ :
1679  $\quad$  Derive  $\backslash ( a + c = 4 \backslash )$  and  $\backslash ( ac = 4 \backslash ).$ 
1680  $\quad$  Solve for  $\backslash ( a \backslash )$  and  $\backslash ( c \backslash )$  to find  $\backslash ( a = c = 2 \backslash ).$ 
1681  $\quad$  But this leads to  $\backslash ( ad + bc = 4 \neq 2 \backslash ),$  a contradiction.
1682 5. **Case  $\backslash ( b = -1 \backslash ), \backslash ( d = -1 \backslash )**$ :
1683  $\quad$  Derive  $\backslash ( a + c = 4 \backslash )$  and  $\backslash ( ac = 8 \backslash ).$ 
1684  $\quad$  Show that no real roots exist for  $\backslash ( a \backslash ),$  so this case is invalid.

```

```

1674
1675 6. **Conclusion**: No valid factorization exists, so  $\langle P(x) \rangle$  is
1676  ↪ irreducible.
1677
1678 ### Lean 4 `have` Statements
1679
1680 ```lean4
1681 theorem exercise_9_4_2c : Irreducible
1682   (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial  $\mathbb{Z}$ ) := by
1683   have h_main : Irreducible (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial
1684   ↪  $\mathbb{Z}$ ) := by sorry
1685   sorry
1686
1687 ```
1688
1689 ### Complete Lean 4 Proof
1690
1691 ```lean4
1692 theorem exercise_9_4_2c : Irreducible
1693   (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial  $\mathbb{Z}$ ) := by
1694   have h_main : Irreducible (X^4 + 4*X^3 + 6*X^2 + 2*X + 1 : Polynomial
1695   ↪  $\mathbb{Z}$ ) := by
1696   -- Assume the polynomial factors into two non-constant polynomials
1697   -- with integer coefficients
1698   refine' (prime_of_irreducible (show Prime (X^4 + 4*X^3 + 6*X^2 + 2*X
1699   ↪ + 1 : Polynomial  $\mathbb{Z}$ ) by
1700   -- Use the fact that the polynomial is prime to show it is
1701   -- irreducible
1702   apply prime_C_mul_X_pow_add_C_of_prime
1703   <;> norm_num
1704   <;>
1705   decide))
1706
1707   <;>
1708   (try norm_num) <;>
1709   (try decide) <;>
1710   (try
1711     {
1712       intro h
1713       -- If the polynomial factors, it must factor into quadratics
1714       rcases h with ⟨p, q, hp, hq, hpq⟩
1715       -- Analyze the possible factorizations
1716       have h1 := congr_arg (fun p => eval 0 p) hpq
1717       have h2 := congr_arg (fun p => eval 1 p) hpq
1718       have h3 := congr_arg (fun p => eval (-1) p) hpq
1719       have h4 := congr_arg (fun p => eval 2 p) hpq
1720       have h5 := congr_arg (fun p => eval (-2) p) hpq
1721       simp at h1 h2 h3 h4 h5
1722       -- Check the possible evaluations to find a contradiction
1723       norm_num [eval_add, eval_mul, eval_pow, eval_C, eval_X, eval_pow,
1724       ↪ eval_mul, eval_add, eval_sub] at h1 h2 h3 h4 h5
1725       <;>
1726       (try { contradiction }) <;>
1727       (try {
1728         have h6 := p.sub_dvd_eval_sub 0 1
1729         have h7 := q.sub_dvd_eval_sub 0 1
1730         have h8 := p.sub_dvd_eval_sub 0 (-1)
1731         have h9 := q.sub_dvd_eval_sub 0 (-1)
1732         have h10 := p.sub_dvd_eval_sub 0 2
1733         have h11 := q.sub_dvd_eval_sub 0 2
1734         have h12 := p.sub_dvd_eval_sub 0 (-2)
1735         have h13 := q.sub_dvd_eval_sub 0 (-2)
1736         simp [eval_add, eval_mul, eval_pow, eval_C, eval_X] at h6 h7 h8
1737         ↪ h9 h10 h11 h12 h13
1738         <;>
1739         (try omega) <;>
1740         (try {
1741           have h14 : p.natDegree ≤ 2 := by

```

```

1728     contrapose! h6
1729     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1730     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1731     ↪ Polynomial.eval_X]
1732     <;>
1733     (try omega) <;>
1734     (try nlinarith)
1735     have h15 : q.natDegree  $\leq$  2 := by
1736     contrapose! h7
1737     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1738     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1739     ↪ Polynomial.eval_X]
1740     <;>
1741     (try omega) <;>
1742     (try nlinarith)
1743     have h16 : p.natDegree  $\leq$  2 := by
1744     contrapose! h8
1745     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1746     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1747     ↪ Polynomial.eval_X]
1748     <;>
1749     (try omega) <;>
1750     (try nlinarith)
1751     have h17 : q.natDegree  $\leq$  2 := by
1752     contrapose! h9
1753     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1754     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1755     ↪ Polynomial.eval_X]
1756     <;>
1757     (try omega) <;>
1758     (try nlinarith)
1759     have h18 : p.natDegree  $\leq$  2 := by
1760     contrapose! h10
1761     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1762     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1763     ↪ Polynomial.eval_X]
1764     <;>
1765     (try omega) <;>
1766     (try nlinarith)
1767     have h19 : q.natDegree  $\leq$  2 := by
1768     contrapose! h11
1769     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1770     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1771     ↪ Polynomial.eval_X]
1772     <;>
1773     (try omega) <;>
1774     (try nlinarith)
1775     have h20 : p.natDegree  $\leq$  2 := by
1776     contrapose! h12
1777     simp_all [Polynomial.eval_add, Polynomial.eval_mul,
1778     ↪ Polynomial.eval_pow, Polynomial.eval_C,
1779     ↪ Polynomial.eval_X]
1780     <;>
1781     (try omega) <;>
1782     (try nlinarith)
1783     -- Now we know the degree of p and q is at most 2
1784     have h22 : p.natDegree  $\leq$  2 := by linarith

```

```

1782 have h23 : q.natDegree  $\leq$  2 := by linarith
1783 have h24 : p = C (p.coeff 0)  $\vee$  p = C (p.coeff 0) * X  $\vee$  p = C
1784  $\leftrightarrow$  (p.coeff 0) * X  $\wedge$  2 := by
1785     have h25 := p.natDegree_le_of_degree_le h22
1786     have h26 := p.natDegree_le_of_degree_le h22
1787     have h27 := p.natDegree_le_of_degree_le h22
1788     cases' lt_or_eq_of_le (natDegree_le_of_degree_le h22) with
1789      $\leftrightarrow$  h28 h28 <;>
1790     simp_all [Polynomial.natDegree_le_iff_degree_le,
1791      $\leftrightarrow$  Polynomial.degree_eq_natDegree]
1792     <;>
1793     (try omega) <;>
1794     (try aesop) <;>
1795     (try {
1796         aesop
1797     }) <;>
1798     (try {
1799         aesop
1800     })
1801 have h25 : q = C (q.coeff 0)  $\vee$  q = C (q.coeff 0) * X  $\vee$  q = C
1802  $\leftrightarrow$  (q.coeff 0) * X  $\wedge$  2 := by
1803     have h26 := q.natDegree_le_of_degree_le h23
1804     have h27 := q.natDegree_le_of_degree_le h23
1805     have h28 := q.natDegree_le_of_degree_le h23
1806     cases' lt_or_eq_of_le (natDegree_le_of_degree_le h23) with
1807      $\leftrightarrow$  h29 h29 <;>
1808     simp_all [Polynomial.natDegree_le_iff_degree_le,
1809      $\leftrightarrow$  Polynomial.degree_eq_natDegree]
1810     <;>
1811     (try omega) <;>
1812     (try aesop) <;>
1813     (try {
1814         aesop
1815     }) <;>
1816     (try {
1817         aesop
1818     })
1819     rcases h24 with (h25 | h25 | h25) <;> rcases h25 with (h26 | h26
1820      $\leftrightarrow$  | h26) <;> simp_all [Polynomial.eval_add,
1821      $\leftrightarrow$  Polynomial.eval_mul, Polynomial.eval_pow,
1822      $\leftrightarrow$  Polynomial.eval_C, Polynomial.eval_X]
1823     <;>
1824     (try omega) <;>
1825     (try nlinarith) <;>
1826     (try {
1827         ring_nf at *
1828         <;>
1829         omega
1830     }) <;>
1831     (try {
1832         norm_num at *
1833         <;>
1834         aesop
1835     })
1836     <;>
1837     aesop

```

```
1836     exact h_main
1837     ...
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
```