BC-Prover: Backward Chaining Prover for Formal Theorem Proving

Yuhang He[1,2,3]*, **Jihai Zhang**[1]*, **Jianzhu Bao**[2,3], **Fangquan Lin**[1],
Cheng Yang[1], **Bing Qin**[2,3], **Ruifeng Xu**[2,3]†, **Wotao Yin**[4]†
[1] DAMO Academy, Alibaba group, Hangzhou, 310023, China
[2] Peng Cheng Laboratory, Shenzhen, China
[3] Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies
[4] DAMO Academy, Alibaba Group US, Bellevue, WA, USA
yuhang.he.hitsz@outlook.com, jianzhubao@gmail.com,
qinb@163.com, xuruifeng.hitsz@gmail.com
{jihai.zjh, fangquan.linfq, charis.yang, wotao.yin}@alibaba-inc.com

Abstract

Despite the remarkable progress made by large language models in mathematical reasoning, interactive theorem proving in formal logic still remains a prominent challenge. Previous methods resort to neural models for proofstep generation and search. However, they suffer from exploring possible proofsteps empirically in a large search space. Besides, they directly use a less rigorous informal proof for proofstep generation, neglecting the incomplete reasoning within. In this paper, we propose BC-Prover, a backward chaining framework guided by pseudo steps. Specifically, BC-Prover prioritizes pseudo steps to proofstep generation. The pseudo steps boost the proof construction in two aspects: (1) Backward Chaining that decomposes the proof into sub-goals for goal-oriented exploration. (2) Step Planning that makes a fine-grained planning to bridge the gap between informal and formal proofs. Experiments on the miniF2F benchmark show significant performance gains by our framework over the state-of-the-art approaches. Our framework is also compatible with existing provers and further improves their performance with the backward chaining technique.

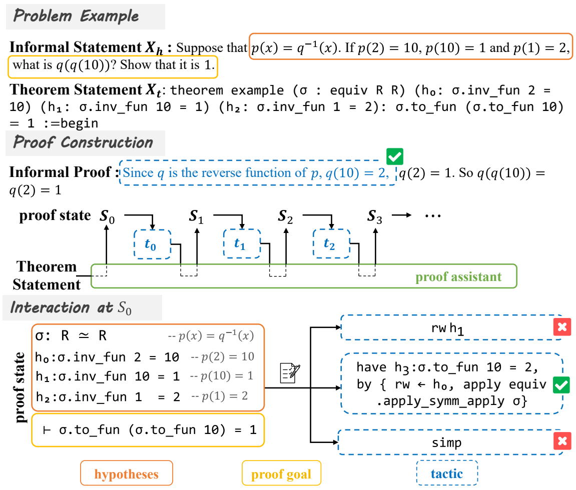


Figure 1: *Problem Example*: An input example form Zheng et al. (2022). *Proof Construction*: A sequence of tactics updates the state interactively to prove the goal. *Interaction at S_0*: Given the value of σ^{-1}, the goal is to prove the value of σ. `rw` fails to rewrite the proof goal with h_1. `simp` fails to simplify the proof goal with existing hypotheses. The `have` tactic successfully proves h_3 reversely by observing the goal. It corresponds to the color sentence in the informal proof.

1 Introduction

Recent years have seen a surge of interest in mathematical reasoning tasks, such as premise selection (Mikula et al., 2023), autoformalization (Zhou et al., 2024a; Wu et al., 2022) and automated theorem proving (Wang et al., 2023b; Azerbayev et al., 2023). Automated theorem proving poses a significant challenge, as it requires the prover to generate validated formal proofs fully automatically in a prohibitively large search space (Lample et al., 2022; Trinh et al., 2024). Therefore, interactive theorem proving (ITP) has emerged as an alternative method for automating theorem proving. ITP typically involves prover writing steps to interact with proof assistants like Isabelle (Nipkow et al., 2002) and Lean (de Moura et al., 2015). Such a process highlights understanding of hypotheses and efficient search strategies to reach the proof goal (Zhang et al., 2023). Besides its huge potential to accelerate research in mathematics, ITP has demonstrated excellent application value in code generation (Polu et al., 2023), synthetic theorem generation (Lin et al., 2024; Huang et al., 2024) and proof refactoring (Zhou et al., 2024b).

As Lean becomes prevalent due to its superiority in interactive mathematical expression, a line of work has explored using language models for programmatically interacting with Lean (Yeh et al.,

*Equal Contribution
† Corresponding Authors

Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pages 3059–3077
November 12-16, 2024 ©2024 Association for Computational Linguistics

2023; Brandfonbrener et al., 2024; Welleck and Saha, 2023). In this paper, we also focus on ITP with Lean. Generally, the ITP task aims at constructing a proof (a sequence of *tactics*) of a proof state (transformed from a theorem statement) as illustrated in Figure 1. *Tactics* are proofsteps for updating the proof state. They must be strictly verified by the *proof assistant* and utilize *hypotheses* to achieve the *proof goal*. Unlike conventional program languages, formal proof language adheres to rigorous mathematical logic, leaving no room for hallucination (Ji et al., 2023). Therefore, the ITP task presents a significant challenge for LLMs.

Existing research on ITP mainly falls into two paradigms: task-specific finetuning and prompting. Task-specific finetuning methods have shown exceptional performance (Han et al., 2022; Wang et al., 2023b). They rely on prohibitive training costs on private datasets, making it impractical in real scenarios without open-source code or models (Polu et al., 2023; Lample et al., 2022). Prompting methods, on the other hand, have already proven to be a powerful copilot in real-world applications (Song et al., 2024). They explore the in-context learning ability of LLMs to infer proofsteps (Thakur et al., 2024; Ying et al., 2024). Our method is also based on the prompting paradigm.

To produce more reliable tactics, most prompting methods involve informal proofs prior to proof construction (Jiang et al., 2023). The informal proofs are solutions in natural language as shown in Figure 1: *Proof Construction*. However, directly applying the informal proofs as in-context examples can mislead the LLMs, as there are gaps between formal and informal proofs. Specifically, informal proofs are often less rigorous and tend to skip steps, making them less reliable. To the best of our knowledge, both prompting and finetuning methods focus on generating forward-chaining tactics, ignoring the backward-chaining strategy. Backward chaining is an inference method described colloquially as working backward from the goal (Huang and Chang, 2023). Ignoring backward chaining could trap the exploration of proofsteps. As shown in Figure 1: *Interaction at* S_0, in order to prove the goal, a hypothesis h_3 should be proved first instead of using existing hypotheses.

In order to alleviate the above problems, we propose BC-Prover, a framework to operate backward-chaining for ITP in Lean. Specifically, given a theorem and its informal statement, our framework first derives an informal proof and pseudo steps. Pseudo steps are specific proof steps further elaborated based on the informal proof, aimed at providing a reference for the formal proof. During proof construction, vanilla provers proposed in previous works generate forward-chaining tactics to interact with Lean(Han et al., 2022; Yang et al., 2023). Instead, inspired by literature in logic reasoning (Poole and Mackworth, 2010), our approach performs (1) **backward chaining**: drawing auxiliary hypotheses about the internal reasoning. As the pseudo steps indicate intermediate steps toward the goal, we employ LLMs to recursively discover provable sub-goals for ultimate proof-finding. To avoid producing misleading steps, our approach performs (2) **step planning**: planning the next proofstep conditioned on the current state. Additionally, it augments the next proofstep with retrieval lemmas and plans similar to Yang et al. (2023). Experiments show that BC-Prover achieves a higher pass rate on the miniF2F benchmark compared with several SOTA baselines. BC-Prover can also collaborate with finetuned models to obtain substantial improvement.

We summarize our contributions as follows:

- We sketch the proof in pseudo steps and make fine-grained step planning to fill the gaps between informal and formal proofs.
- We incorporate a backward-chaining strategy in search of goal-driven tactics to find proof paths efficiently.
- Evaluation on ITP benchmark reveals that our framework outperforms several strong baselines. The backward chaining strategy can also enhance the existing provers.

2 Related Works

2.1 Formal Theorem Proving

Early approaches search proofs in first-order logic automatically (Robinson and Voronkov, 2001; Kovács and Voronkov, 2013). However, the inherent vast search space often limits their practicality in higher-order mathematical problems (Bridge et al., 2014). Later works on theorem proving have thereby focused on ITP paradigm.

With the remarkable achievements of generative models in recent years, task-specific finetuning models are widely used for ITP (Sanchez-Stern et al., 2020; Polu and Sutskever, 2020a). Yang et al.

(2023) implements a retrieval-augmented language model to search appropriate lemmas for tactic generation. Welleck and Saha (2023) makes further improvement by scaling the language model. Recent researches also explore advanced LLMs with sophisticated prompting methods (Poulsen et al., 2024; Yousefzadeh and Cao, 2023). Jiang et al. (2022) integrates LLM and Sledgehammer, a powerful automated prover in Isabelle, for theorem proving. First et al. (2023) investigates LLM for repairing the whole proof after interaction with the proof assistant. Similarly, Thakur et al. (2024) repeatedly corrects proofsteps from assistant feedback. These previous approaches begin with established hypotheses and keep applying tactics in a forward-chaining fashion until the goal is reached. We integrate a backward-chaining strategy to uncover possible facts for proving the goal.

2.2 Proof Autoformulation

The goal of proof autoformalization is to convert informal theorems and proofs into machine-verifiable formats. It has been studied since the early stage of neural models (Kaliszyk et al., 2014). In the era of LLMs, proof autoformulation demonstrates its value in automated theorem proving by translating mathematical statements in natural language into formal proofs (Wang et al., 2020; Wu et al., 2022; Zhao et al., 2023). It promises to facilitate the verification of mathematical papers (Szegedy, 2020). Jiang et al. (2023) introduces a draft-sketch-prove pipeline to formulate informal proof automatically. Subsequent research builds dynamic lemma libraries and achieved substantial improvement (Wang et al., 2023a). Thakur et al. (2024) guides the next tactic generation with informal proof in ITP but only yields a little increment. Despite their contributions, none of the aforementioned methods have investigated pseudo steps as intermediate results toward final proof, nor do they formulate backward-chaining steps in the ITP task.

2.3 Proofstep Generation and Search

Combining language models for proofstep generation and heuristic algorithms for proofstep search has been the key to ITP. A thread of research trains a language model and simply adopts best-first-search (Polu and Sutskever, 2020a; Welleck et al., 2022; Polu et al., 2023; Zheng et al., 2023; Jiang et al., 2021). Subsequent advancements focus on deriving more efficient search strategies like MCTS (Lample et al., 2022). Wang et al. (2023b) adjusts search steps to accommodate proof state complexity, mirroring human reasoning over the entire proof trajectory. Besides, some studies train LLMs on extensive general mathematical corpora and build strong LLM agents for proofstep generation (Shao et al., 2024; Azerbayev et al., 2023; Rozière et al., 2023). These studies are highly relevant to our work, as our goal is to build an LLM agent that uses backward chaining to reduce the search space of forward chaining.

3 Methodology

A problem statement consists of a theorem statement X_t and its informal statement X_h. X_t will be transformed into an initial proof state $S_0 = \{h_1, ..., h_l, g_1, ..., g_m\}$ that holds l hypotheses h and m goals g. A problem example is illustrated in Figure 1: *Problem Example*. The ITP task can be formulated as follows: given X_t, X_h and S_0, a prover needs to generate tactics iteratively to construct a proof. In each iteration, the prover searches for tactics to update the state. The iterations "$S_0 \rightharpoonup S_1 \rightharpoonup \cdots$" finish until all goals are accomplished or the search ends.

3.1 Basic Prover

In general, a basic prover is composed of a proofstep generator and a proofstep search mechanism. The proofstep generator iteratively generates tactics based on the proof state. The proofstep search mechanism controls the overall search process, maintaining states and selecting tactics during proof construction.

Proofstep Generator. Following Polu and Sutskever (2020b), we use a decoder-only language model LM as the proofstep generator. The generator takes a proof state S_i and generates k tactics:

$$\{t_i^0, ..., t_i^k\} = \mathrm{LM}(S_i) \tag{1}$$

Formally, the above process is defined as forward chaining.

Proofstep Search. The goal of the proofstep search is to build a proof tree that incrementally evolves the state through tactic invocations. Starting from the initial state S_0, the prover expands proof states by executing tactics in each iteration. The intermediate states are maintained in a priority queue and expanded based on the cumulative log probability. The cumulative log probability is the summation of the log probabilities of tactics that

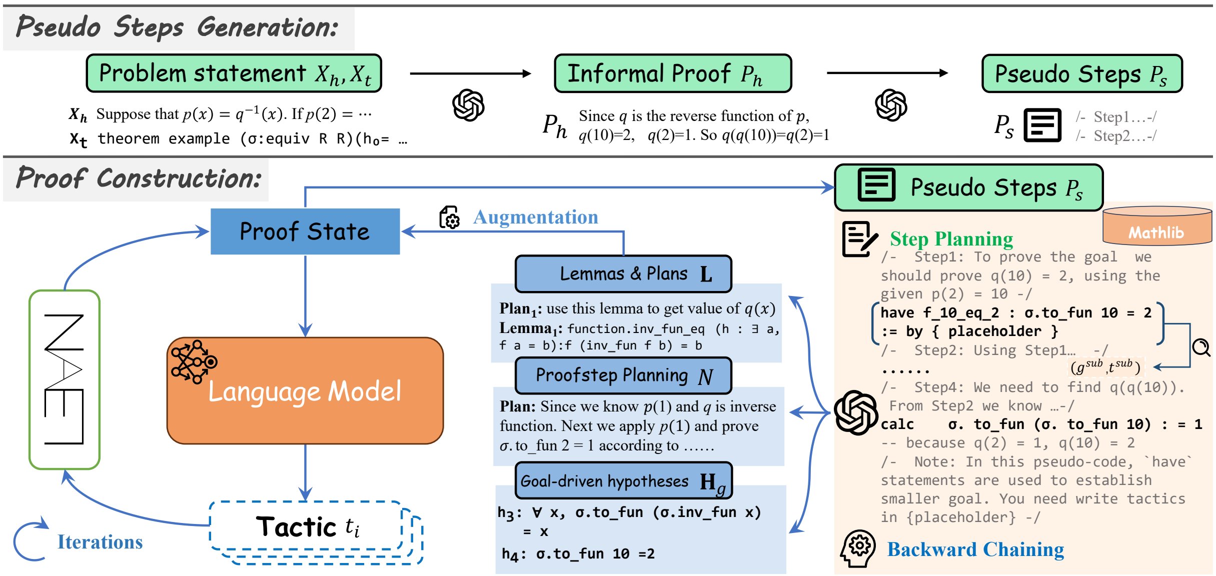


Figure 2: Overview of BC-Prover. It first generates pseudo steps by sketching a proof of the problem. During proof construction BC-Prover iteratively augments the proof state with step planning and backward chaining.

brought us to the next state S_j from S_0. The prover commonly adopts best-first search:

$$S_j = \text{Lean}(t'_{j-1}, S_{j-1}) \tag{2}$$

where Lean is the Lean assistant and S_{j-1} is the current state. The best tactic t'_{j-1} leads to a state with the highest cumulative log probability. It can be regarded that best-first search is operating in a forward-chaining manner.

3.2 BC-Prover

Upon the basic prover, we propose our BC-Prover framework, as shown in Figure 2. BC-Prover first sketches an informal proof and pseudo steps for the input mathematical problem. Subsequently, BC-Prover engages in proof construction through iterative processes. In each iteration, the backward chaining mechanism utilizes the LLM's reasoning ability to discover goal-driven hypotheses. The step planning module derives next-step planning conditioned on the current state. Additionally, BC-Prover retrieves potentially useful lemmas with the help of a retriever and a re-ranking agent. Finally, the proof state is augmented with the aforementioned information for tactic generation.

Pseudo Steps Generation. BC-Prover proceeds pseudo steps generation before proof construction. It formalizes the input problem following recent advances in proof autoformulation (Jiang et al., 2023). Accordingly, the problem is mapped into an informal proof and pseudo steps sequentially:

$$\mathcal{M} : (X_t, X_h) \rightarrow P_h \tag{3}$$

$$\mathcal{M} : (X_t, X_h, P_h) \rightarrow P_s \tag{4}$$

where $\mathcal{M}$ is parameterized by LLM, $\rightarrow$ indicates prompting and parsing procedure to generate the desired results[1]. P_h and P_s denote informal proof and pseudo steps respectively. Pseudo steps are more structured, filling up steps that require explicit proving but are taken for granted in informal proof.

In the following, BC-Prover conducts proof construction. It is guided by the pseudo steps and updates the proof state iteratively. The iterations end when the goals are achieved or the search is finished.

Backward Chaining. Backward chaining starts from the goal and recursively breaks it into sub-goals, which should be asserted as facts for goal achievement (Al-Ajlan, 2015). Analogously, BC-Prover performs backward chaining in each iteration to establish provable hypotheses. The informal proof briefly outlines the proof path. The pseudo steps decompose it into commented proofsteps. As pseudo steps declare necessary sub-goals, our backward chaining is guided by it to establish d sub-goals g^{sub} and corresponding tactics t^{sub}, using LLMs' reasoning ability:

$$\mathcal{M} : (P_s, S_i) \rightarrow \mathbf{H} \tag{5}$$

[1]All specific prompts can be found in Appendix A.1

where $\mathbf{H} = \{(g_0^{sub}, t_0^{sub}), ..., (g_d^{sub}, t_d^{sub})\}$ and the current state S_i is also an essential input since it tells about the existing hypotheses and final goal. Next, all sub-goal and tactic pairs (g^{sub}, t^{sub}) are to be verified by the proof assistant:

$$\mathbf{H}_g = \text{Lean}(\mathbf{H}, S_i) \tag{6}$$

where $\mathbf{H}_g = \{h_0^{goal}, h_1^{goal}, ...\}$. The validated pairs will be introduced as goal-driven hypotheses h^{goal} to augment the proof state:

$$S_i^\theta = [S_i, \mathbf{H}_g] \tag{7}$$

where $[\cdot]$ is the augmentation operation.

Step Planning. Step Planning augments the current state with step-level planning to bridge the gap between informal and formal proofs. Since the proof state involves obscure mathematical notions, we annotate the proof state with a description D. Then an LLM reasons out a next-step planning under the current state. The whole procedure is accomplished by prompting $\mathcal{M}$, denoting as:

$$\mathcal{M} : S_i^\theta \rightarrow D \tag{8}$$

$$\mathcal{M} : (D, S_i^\theta, P_s) \rightarrow N \tag{9}$$

where S_i^θ, P_s, and N are the state, pseudo steps, and planning of the next proofstep respectively. The planning gives instructions on the effective tactics to progress the current state.

Following LeanDojo (Yang et al., 2023), we employ its premise retriever to query a set of lemmas $\mathbf{L}_r$ from mathlib[2]. Afterward, an LLM re-ranking agent selects n lemmas and summarizes plans for their use in the next proofstep (Sun et al., 2023):

$$\mathcal{M} : (\mathbf{L}_r, S_i^\theta, P_s) \rightarrow \mathbf{L} \tag{10}$$

where $\mathbf{L}$ is a set of potentially useful lemmas and relevant brief plans.

Proofstep Generator. Finally, state S_i^θ is augmented with the above-generated results. BC-Prover instruct $\mathcal{M}$ with the proof state S_i^* to generate k tactics:

$$S_i^* = [S_i^\theta, N, \mathbf{L}] \tag{11}$$

$$\mathcal{M} : (S_i^*) \rightarrow \{t_i^0, ..., t_i^k\} \tag{12}$$

Proofstep Search. BC-Prover iteratively guides LLM to predict tactics for the current proof state. The best-first search is impractical since the log probability is inaccessible by calling LLM's API. Inspired by DT-Solver (Wang et al., 2023b), we alternatively select proofsteps based on state complexity. Specifically, the current proof state is expanded with tactic $t_j^{'}$ that leads to the simplest S_j. The simplicity of S_j is measured by the number of tokens. The selected tactic is to interact with Lean as described in Equation 2.

The basic prover searches exhaustively in a forward chaining manner. Our framework, in each iteration, recursively establishes goal-driven hypotheses. Hence, our proofstep search mechanism is actually a bidirectional search, which reduces the search space with backward chaining.

4 Experiment Setup

In this section, we introduce the experiment setup for BC-Prover. Following Polu et al. (2023), we evaluate BC-Prover in Lean. More experimental details and hyperparameters are in Appendix A.

Baselines. Baselines of two paradigms are compared, encompassing the state-of-the-art ITP provers in Lean.

(1) Task-specific finetuning. PACT (Han et al., 2022) co-trains the GPT-f model with nine auxiliary tasks. Expert Iteration (Polu et al., 2023) trains the language model by self-synthetic data from proof searches. ReProver (Yang et al., 2023) proposes a premise-augmented model for theorem proving. LLMStep (Welleck and Saha, 2023) scales the model of ReProver without premise retrieving.

(2) Prompting. CodeLama (Rozière et al., 2023), Deepseek-Math (Shao et al., 2024), LLEMMA (Azerbayev et al., 2023) are LLMs trained on various large-scale mathematical corpus with reinforcement learning technique. We choose their 7B version because 7B models perform the best in their reports. Copra (Thakur et al., 2024) devises a GPT-4-based agent to search and correct tactics from the assistant's feedback. GPT-4 baseline is implemented with `gpt-4-turbo-2024-04 -09` version under few-shot settings like LLEMMA.

Note that we exclude some baselines that are infeasible to compare with (details in Appendix A.2). For example, approaches across different proof assistants are not comparable because of different experiment settings.

Implementation details. In this paper, the LLM is instantiated to be `gpt-4-turbo-2024-04 -09`. In each iteration, BC-Prover generates $k = 16$

[2]Mathlib is a user maintained library for the Lean. It contains a large amount of lemmas for theorem proving

Methods	Search-k	miniF2F-valid	miniF2F-test
Task-specific finetuning			
PACT (Han et al., 2022)	-	23.9%	24.6%
Expert Iteration (Polu et al., 2023)	$\times$64	28.5%	25.9%
ReProver (Yang et al., 2023)	$\times$64	23.8%	26.5%
LLMStep (Welleck and Saha, 2023)	$\times$64	26.2%	27.9%
Prompting			
CodeLama-7B (Rozière et al., 2023)	$\times$32	25.0%	20.5%
Deepseek-Math-7B (Shao et al., 2024)	$\times$32	27.9%	28.3%
LLEMMA-7B (Azerbayev et al., 2023)	$\times$32	26.6%	26.2%
Copra-GPT4 (Thakur et al., 2024)	$\times$60	-	29.9%
GPT-4 (OpenAI, 2023)	$\times$16	22.9%	23.4%
BC-Prover	$\times$16	29.5%	30.7%
BC-ReProver	$\times$16	32.0%	31.6%
BC-LLMStep	$\times$16	35.2%	32.0%
BC-Prover*	$\times$16	**38.9%**	**36.9%**

Table 1: Pass@1 results on the miniF2F benchmark. BC-Prover* denotes the cumulative pass rate of the miniF2F dataset, considering the total number of problems solved using our framework. API cost of our framework is shown in Appendix B.

tactics and $d = 8$ sub-goals. We set $n = 5$ for lemma re-ranking. The temperature is set as 0 in prompting the LLM. We adapt the basic provers into our framework by only augmenting the current state with goal-driven hypotheses. They implement best-first search. More implementation details can be found in Appendix A.3.

Theorem Proving Experimental Setup. In ITP task, we adopt the miniF2F benchmark (Zheng et al., 2022) for comparison with other works. This benchmark contains two split datasets: miniF2F-valid and miniF2F-test, which includes total 488 theorems sourced from Olympiad mathematical problems (AIME, AMC, and IMO) as well as high-school and undergraduate math classes. We follow previous works (Lample et al., 2022) and evaluate on these two splits. We primarily use Lean 3 as the proof assistant. We evaluate the performance using Pass@1 metric: the prover has only one attempt and must find the proof within 100 iterations.

5 Results and Analysis

5.1 Main Result

We present the performance of our framework under two different settings. Table 1 shows the overall performance of the baselines and our proposed model.

BC-Prover Settings. We compare BC-Prover with task-specific finetuning and prompting baselines. Overall, BC-Prover achieves better performance with smaller search-k. Our framework significantly improves over the GPT-4 by at least 6.5% and 7.3% on the miniF2F-valid and miniF2F-test, respectively. BC-Prover outperforms another GPT-4-based Copra with lower search-k on the set and surpasses LLMs of mathematics domains by 10.2% at most. It hints that backward chaining and step planning are important to guide LLMs to produce accurate tactics. Furthermore, task-specific finetuning methods thoroughly explore the search space with 64 search-k while BC-Prover achieves a higher pass rate with 16 search-k. We think it is mainly due to the backward chaining mechanism as analyzed in Section 5.3. In conclusion, BC-Prover presents competitive performance compared with SOTA baselines on the miniF2F benchmark.

Collaborative Settings. Our framework is plug-and-play with task-specific finetuning models. Under collaborative settings, we employ ReProver and LLMStep in Equation 11 to generate tactics. The collaborative models are denoted as BC-ReProver and BC-LLMStep. For a fair comparison with BC-Prover, both BC-ReProver and BC-LLMStep are constrained to only generate 16 tactics. It can be observed that collaborative models performs better than BC-Prover. The possible reason behind this is that LLMs are not particularly trained on Lean. Also, the Lean-related data is scarce and hard to ac-

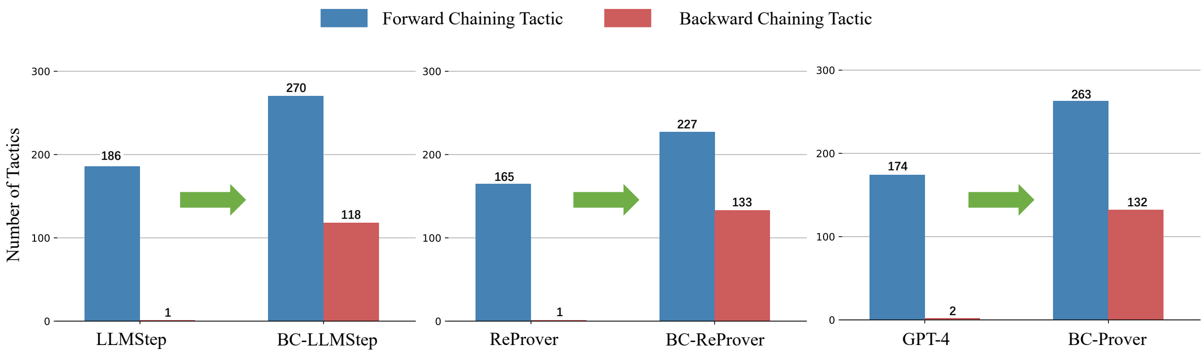


Figure 3: The number of tactics in the constructed proofs. There is an obvious increase in the number of backward chaining tactics after applying our framework.

cess publicly (Han et al., 2022). Besides, both ReProver and LLMStep achieve substantial improvement in collaboration with our framework. On the miniF2F-valid, backward chaining can bring about a 9.0% increase for LLMStep and about 8.2% increase for ReProver. It is worth noting that the collaborative models can reach the final goal with 4 times less search-k. We argue that backward chaining effectively introduces intermediate goals to narrow the search paths for forward chaining.

In line with prior works (Lample et al., 2022), BC-Prover* adds up the number of theorems solved with our framework. In total, our framework successfully carries out 38.9% (95/255) problems on the valid split and 36.9% (90/244) on the test split. The collaboration of our framework and finetuning models is able to discover more new proofs than the original BC-Prover. More experiments and results refer to Appendix D

5.2 Ablation

Methods	miniF2F-valid	miniF2F-test
BC-Prover	29.5%	30.7%
w/o BC	25.4%	25.0%
w/o SP	27.4%	28.3%

Table 2: Ablation Study. Pass@1 results on the miniF2F.

We remove backward chaining (*w/o* BC) and step planning (*w/o* SP) respectively to reveal the effect of each module in our model. The Pass@1 of the miniF2F benchmark is reported in Table 2. All of our proposed modules can bring performance improvements. In particular, applying the BC to our framework contributes about 4.1% and 5.7% pass rate on valid set and test set, respectively, showing the effectiveness of our proposed BC in proof-finding. Also, it can be observed that removing SP also leads to a drop in performance. One big factor is that SP gives comprehensive suggestions on what to do next and how to achieve with extra lemmas, and removing it may exponentially enlarge the forward-searching space.

5.3 Forward vs. Backward Chaining

Forward chaining tactics are machine-validated proofsteps generated by the proofstep generator, while backward chaining tactics (t^{sub}) are proofsteps that draw goal-driven hypotheses for proof construction. To validate the effectiveness of backward chaining, we measure the changes of types of tactics after implementing our framework on the baseline models. From the results in Figure 3, it can be seen that none of the baseline models is able to iteratively decompose the proof goal and create helpful hypotheses, thus limiting their ability to search forward. Also, the number of forward chaining tactics indeed increases with the help of our framework. To be specific, LLMStep generates 84 more forward proofsteps with 118 backward chaining tactics. ReProver, likewise, finds 62 more tactics during proofstep generation. Our LLM-based BC-Prover searches for sug-goals in each iteration, resulting in about 132 useful hypotheses. The above results suggest that backward chaining can steer the search in the correct direction.

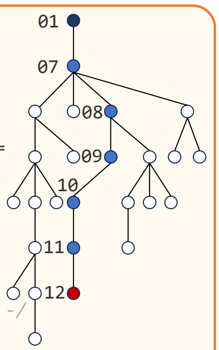
5.4 Search Efficiency Analysis

Methods	Avg.P ↓	Avg.I ↓	Max.I ↓
BC-Prover	1.80	2.30	23
GPT-4	2.07	3.45	61

Table 3: Avg.P indicates the average proofsteps of the proofs. Avg.I and Max.I indicate the average and maximum iterations consumed in finding the proofs, respectively.

GPT-4 Output:

```
01 theorem mathd_algebra_209
02   (σ : equiv ℝ ℝ)
03   (h₀ : σ.inv_fun 2 = 10)
04   (h₁ : σ.inv_fun 10 = 1)
05   (h₂ : σ.inv_fun 1 = 2) :
06   σ.to_fun (σ.to_fun 10) = 1 :=
07 begin
08   norm_num,
09   apply_fun σ.inv_fun at h₁,
10   apply_fun σ at h₂,
11   simp at h₀,
12   sorry,
13   /- search end, goal unproved -/
14 end
```



BC-Prover Output:

```
01 theorem mathd_algebra_209
02   (σ : equiv ℝ ℝ)
03   (h₀ : σ.inv_fun 2 = 10)
04   (h₁ : σ.inv_fun 10 = 1)
05   (h₂ : σ.inv_fun 1 = 2) :
06   σ.to_fun (σ.to_fun 10) = 1 :=
07 begin
08   have f_10eq2: σ.to_fun 10 = 2,
09     by {rw ← h₀, exact σ.right_inv 2},
10   have f_2eq1: σ.to_fun 2 = 1,
11     by {rw ← h₂, exact σ.right_inv 1},
12   have f_f10eqf2:
13      σ.to_fun (σ.to_fun 10) = σ.to_fun 2,
14        by {rw f_10eq2},
15   have id_apply_10_eq_10: id 10 = 10,
16     by {refl},
17   have f_f10eq1:
18      σ.to_fun (σ.to_fun 10) = 1,
19        by {rw [f_f10eqf2, f_2eq1]},
20   exact f_f10eq1,
21 end
```

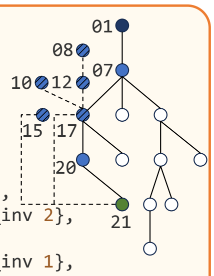


Figure 4: Example outputs and search trees of "theorem mathd_algebra_209". The node is annotated with the line number of the outputs. *Left:* GPT-4 reaches the end of the search and fails to solve the goal. *Right:* BC-Prover reaches the goal with backward chaining, where the dash lines denote the backward chaining route. We highlight the proofsteps of forward chaining: 01-07-08-09-10-11-12 (*left*) and 01-07-17-20-21 (*right*).

To better demonstrate why our proposed framework works, we carry out an analysis of the search efficiency. We compute several metrics from the proof results as shown in Table 3. During the proofsteps construction, the prover tries to find the correct path in the search tree. Avg.P reflects how deep it reaches in the tree. GPT-4 uses up to 61 iterations with 2.07 Avg.P, which means GPT-4 spends most of its efforts in exploring proof paths. Our BC-Prover is lower in Avg.P, Avg.I and Max.I, showing that it can finish searching in 1.80 proofsteps on average and find the correction direction within fewer iterations. In conclusion, BC-Prover exhibits higher search efficiency by incorporating steps planning and backward chaining.

5.5 Case Study

In this section, we present a theorem example that is successfully solved by our framework in Figure 4. More examples are shown in Appendix E. The trees beside the proof are the proof trees in proof constructions. We compare proof from GPT-4 and our BC-Prover. The theorem states the same problem as in Figure 1, where our goal is $\sigma.\texttt{to\_fun}(\sigma.\texttt{to\_fun}\ 10) = 1$.

From Figure 4 *left*, we can see GPT-4 tries to rewrite the existing hypotheses. For example, line 09 rewrites the $\texttt{h}_1$ into $\texttt{h}_1$: $\sigma.\texttt{inv\_fun}(\sigma.\texttt{inv\_fun}\ 10) = \sigma.\texttt{inv\_fun}\ 1$. The transformation makes $\texttt{h}_1$ look like the goal but actually does not drive the search towards the goal. After the tactic in line 11, GPT-4 can not produce any new tactic to update the state. Such a search is purely forward chaining and ends up unsuccessful in 25 iterations (25 nodes in total). In Figure 4 *right*, BC-Prover is guided by pseudo steps and iteratively established sub-goals and tactics. For example, line 08 states a sub-goal $\sigma.\texttt{to\_fun}\ 10 = 2$ and tactic $\texttt{rw} \leftarrow h_0$, $\texttt{exact}\ \sigma.\texttt{right\_inv}\ 2$. Line 08-09 is validated by the Lean and introduce a new hypothesis $\sigma.\texttt{to\_fun}\ 10 = 2$, which is later used in line 12-14. The backward chaining makes the search directly find a fowrad path (01-07-17) close to the final goal. By applying line 20, BC-Prover accomplished the proof with less iteration and proofsteps. We find that the new hypothesis introduced in Line 15 is useless in proving the goal. Similar hypotheses can also be found in other cases as analyzed in Appendix C. How to avoid generating these hypotheses to improve backward chaining is left for future work.

6 Conclusion

In this work, we propose a novel framework, BC-Prover. At the beginning, BC-Prover generates pseudo steps for constructing proofs. During proof construction, BC-Prover operates backward chaining and step planning to construct proofs in formal logic. The backward chaining searches for proofstep in a goal-driven manner. The step planning makes a detailed planning for each proofstep to invoke more accurate tactics. The experiment results on the benchmark demonstrate the superiority of our framework. Our results demonstrate that future work on ITP should incorporate backward chaining to search tactics more effectively.

Limitations

Unlike informal natural language, formal theorems in Lean are rigorous with lots of notions in mathematical language. Although our framework minimizes the gap between informal and formal language, general LLMs are still prone to predict tactics with grammar errors. Unfortunately, BC-Prover outperforms several baselines but fails to solve IMO-level Olympiad problems. Future research could explore devising an LLM agent in the Lean domain to alleviate these problems. During the proof construction, the pseudo steps are not updated as the proof goes on. BC-Prover relies on the self-correction ability of LLM to adjust the proofstep, which could be a weakness for proof construction. The step planning module produces complex instructions. Therefore, it cannot be applied to task-specific fine-tuning models and may degrade LLMs of a small scale (e.g., 7B). We only use ReProver and LLMStep to collaborate with our framework. Consequently, for future research, we aim to evaluate with more finetuning models to verify the effectiveness of the backward chaining technique.

Ethics Statement

We adhere strictly to the licenses and policies of LLMs and publicly available datasets. We follow the usage policy of OpenAI for constructing mathematical proofs and generate no harmful content. The dataset contains mathematical theorems in formal logic and does not involve any ethical problems.

Acknowledgment

This work was supported by Damo Academy through Damo Academy Research Intern Program. This work was partially supported by the National Natural Science Foundation of China 62176076, Natural Science Foundation of GuangDong 2023A1515012922, the Shenzhen Foundational Research Funding JCYJ20220818102415032, the Major Key Project of PCL2021A06, Guangdong Provincial Key Laboratory of Novel Security Intelligence Technologies 2022B1212010005.

References

Ajlan Al-Ajlan. 2015. The comparison between forward and backward chaining. *International Journal of Machine Learning and Computing*, 5(2):106.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *CoRR*, abs/2310.10631.

Haniel Barbosa, Clark W. Barrett, Martin Brain, Gereon Kremer, Hanna Lachnitt, Makai Mann, Abdalrhman Mohamed, Mudathir Mohamed, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Andrew Reynolds, Ying Sheng, Cesare Tinelli, and Yoni Zohar. 2022. cvc5: A versatile and industrial-strength SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer.

David Brandfonbrener, Sibi Raja, Tarun Prasad, Chloe Loughridge, Jianang Yang, Simon Henniger, William E. Byrd, Robert Zinkov, and Nada Amin. 2024. Verified multi-step synthesis using large language models and monte carlo tree search. *CoRR*, abs/2402.08147.

James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. 2014. Machine learning for first-order theorem proving - learning to select a good heuristic. *J. Autom. Reason.*, 53(2):141–172.

Leonardo Mendonça de Moura and Nikolaj S. Bjørner. 2008. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer.

Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The lean theorem prover (system description). In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer.

Emily First, Markus N. Rabe, Talia Ringer, and Yuriy Brun. 2023. Baldur: Whole-proof generation and repair with large language models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 1229–1241. ACM.

Jesse Michael Han, Jason Rute, Yuhuai Wu, Edward W. Ayers, and Stanislas Polu. 2022. Proof artifact co-training for theorem proving with language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jie Huang and Kevin Chen-Chuan Chang. 2023. Towards reasoning in large language models: A survey. In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 1049–1065. Association for Computational Linguistics.

Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo Li, Linqi Song, and Xiaodan Liang. 2024. MUSTARD: mastering uniform synthesis of theorem and proof data. *CoRR*, abs/2402.08957.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Yejin Bang, Andrea Madotto, and Pascale Fung. 2023. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12):248:1–248:38.

Albert Qiaochu Jiang, Wenda Li, Jesse Michael Han, and Yuhuai Wu. 2021. Lisa: Language models of isabelle proofs. In *6th Conference on Artificial Intelligence and Theorem Proving*, pages 378–392.

Albert Qiaochu Jiang, Wenda Li, Szymon Tworkowski, Konrad Czechowski, Tomasz Odrzygózdz, Piotr Milos, Yuhuai Wu, and Mateja Jamnik. 2022. Thor: Wielding hammers to integrate language models and automated theorem provers. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Albert Qiaochu Jiang, Sean Welleck, Jin Peng Zhou, Timothée Lacroix, Jiacheng Liu, Wenda Li, Mateja Jamnik, Guillaume Lample, and Yuhuai Wu. 2023. Draft, sketch, and prove: Guiding formal theorem provers with informal proofs. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Cezary Kaliszyk, Josef Urban, Jirí Vyskocil, and Herman Geuvers. 2014. Developing corpus-based translation methods between informal and formal mathematics: Project description. In *Intelligent Computer Mathematics - International Conference, CICM 2014, Coimbra, Portugal, July 7-11, 2014. Proceedings*, volume 8543 of *Lecture Notes in Computer Science*, pages 435–439. Springer.

Laura Kovács and Andrei Voronkov. 2013. First-order theorem proving and vampire. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer.

Guillaume Lample, Timothée Lacroix, Marie-Anne Lachaux, Aurélien Rodriguez, Amaury Hayat, Thibaut Lavril, Gabriel Ebner, and Xavier Martinet. 2022. Hypertree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Xiaohan Lin, Qingxing Cao, Yinya Huang, Zhicheng YANG, Zhengying Liu, Zhenguo Li, and Xiaodan Liang. 2024. Atg: Benchmarking automated theorem generation for generative language models.

Maciej Mikula, Szymon Antoniak, Szymon Tworkowski, Albert Qiaochu Jiang, Jin Peng Zhou, Christian Szegedy, Lukasz Kucinski, Piotr Milos, and Yuhuai Wu. 2023. Magnushammer: A transformer-based approach to premise selection. *CoRR*, abs/2303.04488.

Tobias Nipkow, Markus Wenzel, and Lawrence C Paulson. 2002. *Isabelle/HOL: a proof assistant for higher-order logic*. Springer.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. 2023. Formal mathematics statement curriculum learning. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Stanislas Polu and Ilya Sutskever. 2020a. Generative language modeling for automated theorem proving. *CoRR*, abs/2009.03393.

Stanislas Polu and Ilya Sutskever. 2020b. Generative language modeling for automated theorem proving. *ArXiv*, abs/2009.03393.

David Poole and Alan K. Mackworth. 2010. *Artificial Intelligence - Foundations of Computational Agents*. Cambridge University Press.

Seth Poulsen, Sami Sarsa, James Prather, Juho Leinonen, Brett A. Becker, Arto Hellas, Paul Denny, and Brent N. Reeves. 2024. Solving proof block problems using large language models. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education, SIGCSE 2024, Volume 1, Portland, OR, USA, March 20-23, 2024*, pages 1063–1069. ACM.

J. Robinson and Andrei Voronkov. 2001. *Handbook of Automated Reasoning: Volume 1*. MIT Press, Cambridge, MA, USA.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

Alex Sanchez-Stern, Yousef Alhessi, Lawrence K. Saul, and Sorin Lerner. 2020. Generating correctness proofs with neural networks. In *Proceedings of the 4th ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2020, London, UK, June 15, 2020*, pages 1–10. ACM.

Stephan Schulz. 2004. System description: E 0.81. In *Automated Reasoning - Second International Joint Conference, IJCAR 2004, Cork, Ireland, July 4-8, 2004, Proceedings*, volume 3097 of *Lecture Notes in Computer Science*, pages 223–228. Springer.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. 2024. Towards large language models as copilots for theorem proving in lean. *CoRR*, abs/2404.12534.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937, Singapore. Association for Computational Linguistics.

Christian Szegedy. 2020. A promising path towards autoformalization and general artificial intelligence. In *Intelligent Computer Mathematics - 13th International Conference, CICM 2020, Bertinoro, Italy, July 26-31, 2020, Proceedings*, volume 12236 of *Lecture Notes in Computer Science*, pages 3–20. Springer.

Amitayush Thakur, George Tsoukalas, Yeming Wen, Jimmy Xin, and Swarat Chaudhuri. 2024. An in-context learning agent for formal theorem-proving.

Trieu Trinh, Yuhuai Tony Wu, Quoc Le, He He, and Thang Luong. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625:476–482.

Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, Heng Liao, and Xiaodan Liang. 2023a. Lego-prover: Neural theorem proving with growing libraries. *CoRR*, abs/2310.00656.

Haiming Wang, Ye Yuan, Zhengying Liu, Jianhao Shen, Yichun Yin, Jing Xiong, Enze Xie, Han Shi, Yujun Li, Lin Li, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2023b. Dt-solver: Automated theorem proving with dynamic-tree sampling guided by proof-level value function. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 12632–12646. Association for Computational Linguistics.

Qingxiang Wang, Chad E. Brown, Cezary Kaliszyk, and Josef Urban. 2020. Exploration of neural machine translation in autoformalization of mathematics in mizar. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, LA, USA, January 20-21, 2020*, pages 85–98. ACM.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023c. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Christoph Weidenbach. 2001. Combining superposition, sorts and splitting. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1965–2013. Elsevier and MIT Press.

Sean Welleck, Jiacheng Liu, Ximing Lu, Hannaneh Hajishirzi, and Yejin Choi. 2022. Naturalprover: Grounded mathematical proof generation with language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Sean Welleck and Rahul Saha. 2023. LLMSTEP: LLM proofstep suggestions in lean. *CoRR*, abs/2310.18457.

Yuhuai Wu, Albert Qiaochu Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J. Prenger, and Animashree Anandkumar. 2023. Leandojo: Theorem proving with retrieval-augmented language models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Eric Yeh, Briland Hitaj, Sam Owre, Maena Quemener, and Natarajan Shankar. 2023. Coprover: A recommender system for proof construction. In *Intelligent Computer Mathematics - 16th International Conference, CICM 2023, Cambridge, UK, September 5-8,* *2023, Proceedings*, volume 14101 of *Lecture Notes in Computer Science*, pages 237–251. Springer.

Huaiyuan Ying, Shuo Zhang, Linyang Li, Zhejian Zhou, Yunfan Shao, Zhaoye Fei, Yichuan Ma, Jiawei Hong, Kuikun Liu, Ziyi Wang, Yudong Wang, Zijian Wu, Shuaibin Li, Fengzhe Zhou, Hongwei Liu, Songyang Zhang, Wenwei Zhang, Hang Yan, Xipeng Qiu, Jiayu Wang, Kai Chen, and Dahua Lin. 2024. Internlm-math: Open math large language models toward verifiable reasoning. *CoRR*, abs/2402.06332.

Roozbeh Yousefzadeh and Xuenan Cao. 2023. Large language models' understanding of math: Source criticism and extrapolation. *CoRR*, abs/2311.07618.

Liao Zhang, Lasse Blaauwbroek, Cezary Kaliszyk, and Josef Urban. 2023. Learning proof transformations and its applications in interactive theorem proving. In *Frontiers of Combining Systems - 14th International Symposium, FroCoS 2023, Prague, Czech Republic, September 20-22, 2023, Proceedings*, volume 14279 of *Lecture Notes in Computer Science*, pages 236–254. Springer.

Xueliang Zhao, Wenda Li, and Lingpeng Kong. 2023. Decomposing the enigma: Subgoal-based demonstration learning for formal theorem proving. *CoRR*, abs/2305.16366.

Chuanyang Zheng, Haiming Wang, Enze Xie, Zhengying Liu, Jiankai Sun, Huajian Xin, Jianhao Shen, Zhenguo Li, and Yu Li. 2023. Lyra: Orchestrating dual correction in automated theorem proving. *CoRR*, abs/2309.15806.

Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. 2022. minif2f: a cross-system benchmark for formal olympiad-level mathematics. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q. Weinberger, and Yuhuai Wu. 2024a. Don't trust: Verify - grounding LLM quantitative reasoning with autoformalization. *CoRR*, abs/2403.18120.

Jin Peng Zhou, Yuhuai Wu, Qiyang Li, and Roger B. Grosse. 2024b. REFACTOR: learning to extract theorems from proofs. *CoRR*, abs/2402.17032.

A More Experiment Setup

A.1 Prompts for BC-Prover

For reproducibility, we provided detailed prompts during the proof construction. Concretely, the prompt we used for generating the pseudo steps (Equation 3 and 4) is in Figure 5 and 6. The prompt for backward chaining (Equation 5) is in Figure 7. The prompt for generating next-step planning (Equation 8 and 9) is in Figure 8 and 9. The prompt for summarizing plans for lemma usages (Equation 10) is in Figure 10. The prompt for generating proofsteps in forward chaining (Equation 11) is in Figure 11.

In particular, we demonstrate the sample outputs of backward chaining, next-step planning, and plans for lemma usages in Figure 13, 14 and 15, respectively.

```
As a mathematician and expert, your task is to provide a correct, concise, and clear mathematical answer according to the theorem and its informal statement. Note that the theorem is provable.
{examples}
## Formal theorem:
{theorem_statement}
## Informal statement:
{informal_statement}
## Informal proof:
```

Figure 5: The prompt for generating informal proof.

```
As a mathematician and expert in Lean theorem prover, your task is to write a pseudo code in Lean 3 in response to a problem statement. Your pseudo code should be structured and clearly written, meeting the following criteria:
- It is readable and must be broken down into numerical steps like 'Step1', 'Step2' ...
- Steps of the proof should be explained in detail using comments enclosed in '/-' and '-/'
- Be clear and concise, avoiding any unnecessary or apologetic language.
- Make sure each step of the pseudo-code can be easily converted into formal Lean 3 code.
- Please use NO 'sorry' tactic or placeholders for proofs or assumptions in the pseudo-code.
- Assume you have already imported the necessary Mathematic Library to finish the proof.
## informal problem statement:
{informal_statement}
## informal proof of the problem:
{informal_proof}
Please wirte the pseudo code:
{theorem_statement}
PSEUDO-CODE:
```

Figure 6: The prompt for generating pseudo steps.

A.2 Justification for Excluding Baselines

In Table 1, we compare BC-Prover with recently released LLM in mathematics to our knowledge. We empirically excluded three task-specific finetuned provers targeting ITP in Lean. Here, we focus the discussion on reasons for excluding the three provers (Lample et al., 2022; Polu and Sutskever, 2020b; Polu et al., 2023). Most importantly, it is infeasible to reproduce their work with reasonable effort because they didn't release any code, pre-trained models, or available training datasets. Therefore, we can only compare the Pass@1 metrics reported in their papers. However, it is also impractical due to several reasons:

```
As a mathematician and expert in Lean theorem prover, your task is to analyze the pseudo-code. Please consider what is missing to decompose and achieve the proof goal in a backward reasoning manner. Request useful and reusable hypotheses, meeting the following criteria:
- The hypotheses should be created with Structured Tactic Proofs. That is they should be introduced with the tactic 'let', 'have', or 'suffices'(it adds the hypothesis to the current goal).
- Please make sure that the hypotheses are valid in Lean and you have to successfully proof them in the format: 'have hypo: (sub-goal_statement), by {{<valid_proof>}},'.
- You can try smart tactics to prove the hypothesis like 'ring'(prove equalities in commutative rings) and 'nlinarith'(handles some goals in nonlinear arithmetic).
- The hypotheses should be non-trivial and able to cover a large step in proofs. You can refer to the pseudo-code for inspiration.
- Please use NO placeholders for proofs or assumptions in the <valid_proof>.
- DO NOT generate hypotheses that already exist in the proof state.
- DO NOT generate the keyword 'sorry'.
## Pseudo-code:
{pseudo_steps}
## Please provide {k} effective hypotheses according to the Current Proof State:
{current_state}
#1
“‘lean
have hypothsis1: (<sub-goal_statement>), by {{<tactics>}},
“‘
#2
```

Figure 7: The prompt for backward chaining.

- Lample et al. (2022) only report the Pass@64 on the miniF2F benchmark with their approach. They constructed a synthetic training dataset named Equations, which is not publicly available. Their model is also inaccessible. As a result, we cannot make a fair comparison or reproduce their work.
- Polu and Sutskever (2020b) didn't report their result on the miniF2F benchmark because they published their works before the miniF2F was officially released. They didn't make their model public so it is also difficult to reproduce their work.
- Polu et al. (2023) further fine-tuned their models on new proofs collected through their interaction with Lean on miniF2F benchmark which is supposed to be used in the evaluation. They also construct extra proofs to transfer to miniF2F. Considering the potential overfitting issue, we compare with a generalized model result reported in their paper (Expert Iteration in Table 1).
- We would like to evaluate them under our framework to verify the backward chaining mechanism, whereas none of them make the models public. Hence, we only use ReProver and LLMStep to collaborate with our framework.

```
As a mathematician and expert in Lean theorem prover, your task is to shortly explain the current proof state. Your explanation should:
- Be clear and concise, avoiding any unnecessary or apologetic language.
- Only briefly state the existing hypotheses and the goal.
- There is no need to provide an approach or solution.
## Current State:
{proof_state}
```

Figure 8: The prompt for generating proof state description.

```
As a mathematician and expert in Lean theorem prover, your task is to derive exactly one next proof step according to the current state, meeting the following criteria:
- You should analyze the state and align the next step with one step in the pseudo-code.
- If the aligned target in the pseudo-code is missing or unprecise, the predicted next step should be further decomposed.
- The predicted next proof step must be close to Lean code.
- Do not fabricate hypotheses or lemmas that didn't appear in the context.
- Please use NO 'sorry' or placeholders for proofs and assumptions.
## Pseudo Code:
{pseudo_steps}
## Current State:
{proof_state}
{description}
## The predicted step should be clear, concise, and easy to translate into Lean code:
```

Figure 9: The prompt for generating next-step planning.

Besides formal theorem proving in Lean, we notice that recent researchers also target other proof assistants like Isabelle, Coq, Holight, and Metamath. Unfortunately, it is generally impractical to compare with their work for mainly three reasons:

- Different proof assistants have different characteristics. As for Isabelle, it has powerful automatic reasoning tools like Sledgehammer. Sledgehammer integrates automated theorem provers (ATPs) into Isabelle environment. When Sledgehammer is called, it will try to prove the conjecture using strong, external ATPs like E (Schulz, 2004), SPASS (Weidenbach, 2001), Vampire (Kovács and Voronkov, 2013), Z3 (de Moura and Bjørner, 2008), or cvc5 (Barbosa et al., 2022). And yet, Lean does not have equally powerful tools. Many ITP frameworks in Isabelle make use of this characteristic and achieve higher scores in miniF2F benchmark.
- Different evaluation metrics. Most studies on ITP in Isabelle evaluate their approaches with the pass rate within 100 or 200 attempts(Wang et al., 2023a; Zheng et al., 2023). Our framework, however, adopts Pass@1 as the metric.
- Use of human-verified informal proof. Jiang et al. (2023) and Wu et al. (2022) use human-

As a mathematician and expert in Lean theorem prover, your task is to recall the lemmas in the mathlib3 library and find at least {k} most helpful lemmas for theorem proving. You are required to both analyze the pseudo-code and select appropriate premises from the premise list, meeting the following criteria:
- The pseudo-code provides valuable information about which lemma should be selected.
- Please only select lemmas that help solve the current proof state.
- You should think step by step: make a brief analysis in one single sentence and select the lemma.
- Your response should be clear and concise, ignoring the useless lemmas:
"'text
Analysis_1: <analysis_holder>
Selection_1: <selectioin_holder>
Analysis_2: <analysis_holder>
Selection_2: <selection_holder>"'
##Pseudo-code:
{pseudo_steps}
##Lemma List:
{retrieved_lemmas}
##Proof State:
{proof_state}
Please select at least {k} useful lemmas in the Lemma List or the mathlib3 library.

Figure 10: The prompt for summarizing plans for lemma usages.

As a mathematician and expert in Lean 3 theorem prover, your task is to analyze the current proof state (telling you what facts you have already established and what goals remain to prove) and given theorem (including the pseudo-code, informal statement, and some potentially useful lemmas). You should provide {k} tactic(s) helpful toward proving the proof state, meeting the following criteria:
- Pseudo-code gives valuable hints. The generated tactic should STRICTLY correspond to steps in the pseudo-code.
- Assume you have already imported all lemmas from mathlib3 libraries to finish the proof. You are strongly encouraged to wisely 'apply', 'exact', or 'rw' with lemmas to solve the proof state.
- You are encouraged to decompose and reduce the proof state using 'rewrite', 'field_simp', 'let'.
- 'revert' (move hypotheses into the goal and yield an implication) and 'intros'/'simp_intros' (inverse to 'revert') some hypotheses are helpful to restructure and simplify the state.
- Each tactic should be explained in detail using comments enclosed in '/-' and '-/'.
- Please use NO placeholders for proofs or assumptions in the tactic.
- DO NOT use the keyword 'sorry' in your tactic.
- Do not fabricate hypotheses that didn't appear in the context window.
Potentially Useful Lemmas from mathlib3 (For example):
{lemmas_plans}
Pseudo Code:
{next_step_planning}
Do not change the current proof state because you are only focusing on solving the current problem. Please provide at least {k} effective and commented tactic(s) according to the pseudo-code. Restate the current state before the tactic
{augmented_proof_state}

Figure 11: The prompt for generating proofsteps in forward chaining.

verified informal proofs to refactor ITP into translation between informal and formal proof. In real-world mathematics proving, only the problem is given. Our framework focuses on more realistic scenarios.

Owing to the above-mentioned reasons, we only focus on comparing baselines targeting solving the Lean problem on the miniF2F benchmark similar with Yang et al. (2023).

A.3 More Implementation Details

In this section, we provide more implementation details of our framework.

(1) How do we implement backward chaining? By definition, backward chaining starts from the goal and recursively breaks it into sub-goals, which should be asserted as facts for goal achievement (Al-Ajlan, 2015). The key is to generate a Lean format statement that opens up a sub-goal and state tactics to prove the sub-goal. We implement Equation 5 using structured tactic proofs in Lean [3]. In particular, we define backward chaining as have statements in structured tactic proofs: `have` h^{goal} : g^{sub} `by {` t^{sub} `}`. The LLM is required to generate `have` statements. After verifying the statement, we can introduce the goal-driven hypothesis h^{goal} by interacting with Lean.

(2) How do we parse the responses from LLM? We officially set $k = 16$ in forward chaining and $d = 8$ in backward chaining. Occasionally, we may end up getting less or more than the predefined amount because the generative models could not strictly follow the input instructions. In such cases, we cut off the results to make sure we get amount less than k or d.

(3) How does our framework collaborate with task-specific finetuning models? Task-specific finetuning models are finetuned on sequences of the form:

```
[s]proof-state[PROOFSTEP]tactic[/s]
```

where the proof state consists of hypotheses and proof goal. Augmenting the proof state with N and $\mathbf{L}$ is infeasible. Therefore, we only reconstructed the proof state with $\mathbf{H}_g$. Besides, their input context is limited so we only reconstructed the proof state in the first 10 iterations to avoid exceeding the maximum input length.

B Computational Cost

We record the total computational cost of calling `gpt-4-turbo- 2024-04-09` API for solving the miniF2F benchmark in one attempt. The estimated token and dollar cost are shown in Table 4. It is worth noting that BC-Prover is fully based on LLM so it calls the API many more times than BC-Prover and BC-LLMStep.

[3] https://leanprover.github.io/theorem_proving_in_lean/propositions_and_proofs.html#introducing-auxiliary-subgoals

Methods	input tokens	output tokens	cost
BC-Prover	2.1M	0.7M	45.13$
BC-ReProver	0.6M	0.3M	15.92$
BC-LLMStep	0.6M	0.3M	16.09$

Table 4: The input and output token in million. The API cost in dollar.

C Quantitative Analysis of Backward Chaining

```
   simpl_mult_eighteen: (18 * 18) % 10 = 4
   lemma_div_1529_6: 1529 / 6 = 254
👍 h₇ : g (f 5 - 1) = g 6
   h₁ : (3 : ℝ) ≠ 0
👍 nonneg_of_sq_diff : ∀ (a b : ℝ), 0 ≤ (a - b - 1) ^ 2
   h₂ : 2 * (a + b) = 4
   h_x_eq_3y : x = 3 * y,
   hypothesis1: 1 % 10 = 1
👍 lemma_completing_the_square: ∀ x : ℝ, x^2 - 14 * x + 3 = (x - 7)^2 - 46
👍 h₁ : x^2 - 5*x - 14 ≤ 0
👍 base_and_units: ∀ a : ℕ, ∃ b u : ℕ, a = 10 * b + u ∧ u < 10
   basic_ineq: ∀ (m n : ℕ), 0 < n -> m * 1 ≤ m + 1
   h_sq_neg4_mod_17 : ((-4 : ℤ)^2 % 17) = 16 % 17
👍 hypothesis1: ∀ x, σ.to_fun (σ.inv_fun x) = x
```

Figure 12: We consider hypotheses that contribute little to a proof goal to be of low quality.

Although our framework invokes the backward chaining in mathematical proving, we still wonder what else can be done to further prompt ITP. We conduct a quality analysis of our hypotheses produced by backward chaining. First of all, we sample around 200 hypotheses from the constructed proofs. Some examples are listed in Figure 12. We found that most of them are not of high quality. They are either trivial or rather similar to existing lemmas. This sort of hypothesis might contribute little to finding the proofs. Furthermore, we estimated the number of reusable hypotheses by whether they are invoked throughout the whole proof. We found out that only around 16% tactics make a difference in proof-finding. Even so, our framework managed to solve some of the complex problems and cause no performance degradation. In conclusion, it still remains a problem how to work in backward chaining to generate high-quality hypotheses, which is left for future work.

D More Experiment Result

Autoformulation Settings. Autoformalization is the task of automatically translating natural language mathematics into a formal language that can be verified by a program. It requires a profound understanding of semantics across informal and formal mathematics. Here, we evaluate BC-Prover in proof autoformalization task settings, where a correct human-written informal proof is given. As results in Table 5 illustrate, BC-Prover can find more proofs in miniF2F-test. Concretely, the Pass@1 improves by 3.3% on the miniF2F-test. It implies that LLM still struggles to generate a correct informal proof. A possible solution is to vote for the best informal proof from multiple candidates like self-consistency (Wang et al., 2023c), which we plan to study in future work.

Methods	miniF2F-valid	miniF2F-test
BC-Prover	29.5%	30.7%
BC-Prover + human informal	29.9%	34.0%

Table 5

Category of Solved Proofs. The MiniF2F benchmark contains 488 problems in various categories. The problems can be categorized into Olympiad problems (AIME, AMC, and IMO), number theory problems, algebra problems, and induction problems. Among them, the Olympiad problems are the most difficult. In total, we find proofs of 97 algebra problems, 67 number theory problems, 21 Olympiad problems (19 AMC, 2 AIME, 0 IMO), and 0 induction problems.

E More Cases

Here, we present more proofs found by our framework in Figure 16, 17 and 18. The informal problem statements and human-written informal proofs are displayed for readers to understand.

Input

As a mathematician and expert in Lean theorem prover, your task is to analyze the pseudo-code. Please consider what is missing to decompose and achieve the proof goal in a backward reasoning manner. Request useful and reusable hypotheses, meeting the following criteria:
- The hypotheses should be created with Structured Tactic Proofs. That is they should be introduced with the tactic ‘let‘, ‘have‘, or ‘suffices‘(it adds the hypothesis to the current goal).
- Please make sure that the hypotheses are valid in Lean and you have to successfully proof them in the format: ‘have hypo: (sub-goal_statement), by {{<valid_proof>}},‘.
- You can try smart tactics to prove the hypothesis like ‘ring‘(prove equalities in commutative rings) and ‘nlinarith‘(handles some goals in nonlinear arithmetic).
- The hypotheses should be non-trivial and able to cover a large step in proofs. You can refer to the pseudo-code for inspiration.
- Please use NO placeholders for proofs or assumptions in the <valid_proof>.
- DO NOT generate hypotheses that already exist in the proof state.
- DO NOT generate the keyword ‘sorry‘.

Pseudo-code:
{pseudo_steps}
Please provide {k} effective hypotheses according to the Current Proof State:
n : N,
h : n % 7 = 5
⊢ 5 * n % 7 = 4
#1
```lean
have hypothsis1: (<sub-goal_statement>), by {{<tactics>}},
```
#2

Output

#2
```lean
have h_mul_mod : 5 * 5 % 7 = (5 % 7) * (5 % 7) % 7, from nat.mul_mod 5 5 7,
```
#3
```lean
have h_25_mod_7 : 25 % 7 = 4, by norm_num,
```
#4
```lean
have h_mul_mod : ∀ a b n : N, (a * b) % n = ((a % n) * (b % n)) % n, from nat.mul_mod,
......
```

Figure 13: Prompt example for sub-goal and tactic pairs generation in Backward Chaining

Input

As a mathematician and expert in Lean theorem prover, your task is to recall the lemmas in the mathlib3 library and find at least {k} most helpful lemmas for theorem proving. You are required to both analyze the pseudo-code and select appropriate premises from the premise list, meeting the following criteria:

- The pseudo-code provides valuable information about which lemma should be selected.
- Please only select lemmas that help solve the current proof state.
- You should think step by step: make a brief analysis in one single sentence and select the lemma.
- Your response should be clear and concise, ignoring the useless lemmas:

“‘text
Analysis_1: <analysis_holder>
Selection_1: <selectioin_holder>
Analysis_2: <analysis_holder>
Selection_2: <selection_holder>“‘
Pseudo-code:
{pseudo_steps}
Lemma List:
[Premise_0] lemma <a>nat.mul_mod (a b n : N) : (a * b) % n = ((a % n) * (b % n)) % n
[Premise_1] lemma <a>nat.add_mod (a b n : N) : (a + b) % n = ((a % n) + (b % n)) % n
[Premise_2] lemma <a>nat.mod_add_div’ (m k : N) : m % k + (m / k) * k = m
[Premise_3] ......
Proof State:
{proof_state}
Please select at least {k} useful lemmas in the Lemma List or the mathlib3 library.

Output

Selection_1: none required directly, as the hypothesis itself provides the needed relationship, so we move on to implications of the hypothesis.
Analysis_2: The transition from modular equivalence to a statement about the modulus operation’s result is required to conclude the proof. We need a lemma that directly allows converting a "less than" relationship into an equality under modulus.
Selection_2: <a>nat.mod_eq_of_lt - This lemma can be used to convert the modular equivalence between ((5 * n) mod 7) and 4 into an equality statement, assuming we have established that $5 * n < 7$ which isn’t directly required here but is implied in the kind of transition we are making.
......
Therefore potentially useful premises are:
[nat.mul_mod] lemma <a>nat.mul_mod (a b n : N) : (a * b) % n = ((a % n) * (b % n)) % n
[nat.mod_eq_of_lt] lemma <a>nat.mod_eq_of_lt a b : nat (h : a < b) : a % b = a
......

Figure 14: Prompt example for lemmas and plans generation in Step Planning

Input

As a mathematician and expert in Lean theorem prover, your task is to derive exactly one next proof step according to the current state, meeting the following criteria:

- You should analyze the state and align the next step with one step in the pseudo-code.
- If the aligned target in the pseudo-code is missing or unprecise, the predicted next step should be further decomposed.
- The predicted next proof step must be close to Lean code.
- Do not fabricate hypotheses or lemmas that didn't appear in the context.
- Please use NO 'sorry' or placeholders for proofs and assumptions.

Pseudo Code:
{pseudo_steps}
Current State:
n : $\mathbb{N}$, h : n % 7 = 5, h_mod_25_4 : 25 % 7 = 4, h_5_mod_7 : 5 % 7 = 5, h_final_mod : 5 % 7 * 5 % 7 = 4, h_5_mul_5_mod_7 : 5 * 5 % 7 = 4, h_mul_mod : ∀ (a b n : $\mathbb{N}$), a * b % n = a % n * (b % n) % n
⊢ 5 % 7 * n % 7 = 4
{description}
The predicted step should be clear, concise, and easy to translate into Lean code:

Output

Given the current state and the hypotheses provided, we observe that the hypotheses.

......

So, the predicted Lean code as the next logical step would be to apply 'h_mul_mod' using our values '5', 'n', and '7', followed by substituting 'n % 7' with '5', guided by 'h'.

```lean
have h_step : 5 % 7 * n % 7 % 7 = 5 * 5 % 7, from eq.trans (h_mul_mod 5 n 7) (by rw [h, h_5_mod_7]),
```

This step uses 'eq.trans' to transform the application of 'h_mul_mod' into an intermediate step towards our goal by rewriting 'n % 7' as '5', utilizing 'h' and recognizing the modulo property of '5' as per 'h_5_mod_7'. The application of 'by rw [h, h_5_mod_7]' rewrites the expression according to our hypothesis 'h' and the property of modulo '5' modulo '7'.

Figure 15: Prompt example for next proofstep planning in Step Planning

```
theorem amc12a_2009_p5
  (x : ℝ)
  (h₀ : x^3 - (x + 1) * (x - 1) * x = 5) :
  x^3 = 125 :=
begin
  have h_cube_pos : 0 ≤ x ^ 3,
    by {  apply pow_nonneg,  linarith,},
  have h_five_pos : 0 ≤ (5 : ℝ), by {  norm_num,},
  have h_125 : 5 ^ 3 = 125, by {  norm_num,},
    revert h₀
    ring
    intro h_x_eq_5
    revert x,
    simp [h_125],
  have h_final: ∀ (x : ℝ), x = 5 → x^3 = 125,
    by { intros x hx, rw hx, norm_num, },
  intro x,
  intros h₀ h₁,
  exact h_final x h₁,
end
```

```
informal_statement:
One dimension of a cube is increased by $1$, another is decreased by
$1$, and the third is left unchanged. The volume of the new
rectangular solid is $5$ less than that of the cube. What was the
volume of the cube?
$\textbf{(A)}\ 8 \qquad \textbf{(B)}\ 27 \qquad \textbf{(C)}\ 64
\qquad \textbf{(D)}\ 125 \qquad \textbf{(E)}\ 216$
Show that it is \text{(D)}.
```

```
informal_proof:
Let the original cube have edge length $a$. Then its volume is $a^3$.

The new box has dimensions $a-1$, $a$, and $a+1$, hence its volume is
$(a-1)a(a+1) = a^3-a$.

The difference between the two volumes is $a$. As we are given that
the difference is $5$, we have $a=5$, and the volume of the original
cube was $5^3 = 125\Rightarrow\text{(D)}$.
```

Figure 16: An example proof of BC-Prover.

```
theorem mathd_algebra_246
  (a b : ℝ)
  (f : ℝ → ℝ)
  (h₀ : ∀ x, f x = a * x^4 - b * x^2 + x + 5)
  (h₂ : f (-3) = 2) :
  f 3 = 8 :=
begin
  have h₃ : f (-3) = a * (-3)^4 - b * (-3)^2 - 3 + 5,
    from h₀ (-3),
  have h₅ : 81*a - 9*b = 0, by linarith,
  have h₆ : f 3 = a * 3^4 - b * 3^2 + 3 + 5, from h₀ 3,
  have h₇ : a * 3^4 = 81*a, by ring,
  have h₈ : b * 3^2 = 9*b, by ring,
  have h₁₂ : 3 + 5 = 8, by ring,
  nlinarith,
end
```

```
informal_statement:
If $f(x)=ax^4-bx^2+x+5$ and $f(-3)=2,$ then what is the value of
$f(3)$? Show that it is 8.
informal_proof:
Evaluating $f(x)$ for $x=3$ and $x=-3$, we have
\[\left\{ \begin{aligned} f(3)& = a \cdot 3^4 - b \cdot 3^2 + 3 + 5,
\\ f(-3) &= a \cdot (-3)^4 - b \cdot (-3)^2 + (-3) + 5. \end{aligned}
\right.\]If we subtract the second equation from the first equation,
all the terms but one cancel out, and we get \[f(3) - f(-3) = 3 - (-3)
= 6.\]Thus, if $f(-3) = 2,$ then $f(3) = f(-3) + 6 = 2 + 6 = 8.$
```

Figure 17: An example proof of BC-LLMStep.

```
theorem algebra_apb4leq8ta4pb4
  (a b : ℝ)
  (h₀ : 0 < a ∧ 0 < b) :
  (a + b)^4 ≤ 8 * (a^4 + b^4) :=
begin
  have hypothesis4: 0 ≤ (a - b)^2,
       by {  exact pow_two_nonneg (a - b),},
  have hypothesis6: 2 * (a^2 * b^2) ≤ a^4 + b^4, by
    {
      have h₁: 0 ≤ (a^2 - b^2)^2,
       from pow_two_nonneg (a^2 - b^2),
      simp [pow_two] at h₁,
      nlinarith,
    },
  have hypothesis7: 6 * a^2 * b^2 ≤ 8 * a^2 * b^2,
       by { nlinarith,},
  nlinarith,
end
```

```
informal_statement:
Show that for any positive real numbers $a$ and $b$, $(a+b)^4 \leq
8(a^4 + b^4)$.
informal_proof:
Re-arranging, we must prove $(\frac{a+b}{2})^4\leq\frac{a^4 + b^4}{2}$.
We prove the more general statement $(\frac{a+b}{2})^n\leq\frac{a^n +
b^n}{2}$ for integers $n > 0$ by induction.

The result is trivial for $n=1$. Let us assume the property holds for
$n \geq 1$.

We have that $\left(\frac{a+b}{2}\right)^{n+1} =
\left(\frac{a+b}{2}\right)^n \frac{a+b}{2} \leq \frac{a^n+b^n}{2}
\frac{a+b}{2}$

However, $\frac{a^{n+1}+b^{n+1}}{2} - \frac{a^n+b^n}{2} \frac{a+b}{2}
= \frac{(a^n - b^n)(a-b)}{4}$.

$a^n - b^n$ and $a-b$ have the same sign so $\frac{(a^n - b^n)(a-b)}{4}
\geq 0$ and $\frac{(a^n - b^n)(a-b)}{4} \geq 0$.

As a result, $\left(\frac{a+b}{2}\right)^{n+1} \leq
\frac{a^{n+1}+b^{n+1}}{2}$ and the property holds in $n+1$.

By induction, the result is true for any natural number $n \geq 1$.
```

Figure 18: An example proof of BC-ReProver.