

# 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 LOC-DECOMP: LLM AUTOFORMALIZATION VIA LOGICAL CONCEPT DECOMPOSITION AND ITERATIVE FEEDBACK CORRECTION

Anonymous authors

Paper under double-blind review

## ABSTRACT

Autoformalization—the process of converting natural language mathematical statements into machine-verifiable formal code—plays a critical role in ensuring the reliability of mathematical reasoning generated by large language models (LLMs). Recent studies show that LLMs exhibit strong potential in automating this process, producing formal code for systems such as Lean 4, Coq, and Isabelle. Despite prominent advances, existing LLM-based autoformalization methods remain limited: they lack the ability to provide reliable semantic consistency checks to ensure that the formal code accurately preserves the meaning of the original statement. Furthermore, such methods are unable to support iterative improvement through corrective feedback. To address these limitations, we propose Loc-Decomp, a novel framework that integrates an automatic semantic consistency checker and the Lean 4 compiler to iteratively refine LLM-generated formalizations, ensuring both semantic consistency and syntactic correctness. Our approach introduces three key innovations: **(1)** A structured and COT-like formalization template that decomposes complex formalization tasks into modular, foundational components, and systematically assembles them—like building blocks—into a complete formal expression. **(2)** A semantic self-checking mechanism based on a divide-conquer-merge strategy to detect subtle inconsistencies between the formalization and the original statement. **(3)** An iterative feedback-driven refinement loop that leverages both semantic and syntactic error signals to guide the LLM in progressively improving the formal output. By integrating these innovations, Loc-Decomp significantly enhances the accuracy of LLM-driven formalization, reduces reliance on human intervention, and moves closer to truly reliable automated reasoning. [Extensive experiments on high-school-level and undergraduate-level datasets demonstrate that our approach achieves a significantly higher formalization success rate compared to baseline methods and state-of-the-art \(SOTA\) models. On the PutnamBench dataset, for instance, our method attains a success rate of 93.09%, representing an improvement of 18 percentage points over the previous SOTA SFT-based model.](#)

## 1 INTRODUCTION

Statement formalization(Weng et al., 2025; Gonthier, 2007; Szegedy, 2020) denotes the process of converting a mathematical statement into a formal language—such as Lean 4, Coq, or Isabelle (de Moura et al., 2015; Bertot & Castéran, 2004; Paulson, 1994)—which constitutes a necessary step for the verification of mathematical reasoning in theorem provers(Zhou et al., 2024). A successful formalization must not only satisfy syntactic correctness, as verified by compiler checks, but also ensure semantic consistency by faithfully preserving the meaning of the original statement. However, this task is widely recognized as highly labor-intensive, owing to the inherent flexibility and ambiguity of natural language, which pose significant challenges to automation (Yang et al., 2024).

Recent studies leveraging large language models (LLMs) (Achiam et al., 2023; Team et al., 2023; Liu et al., 2024; Azerbayev et al., 2023b) for autoformalization have shown promising progress and achieved notable results on relatively simple statements. These approaches include both prompt engineering with candidate scoring using general-purpose LLMs and methods based on supervised

054 fine-tuning on domain-specific datasets (Li et al., 2024a; Ying et al., 2024). However, when dealing  
 055 with more complex mathematical statements—such as those in probability, combinatorics, or  
 056 geometry (Trinh et al., 2024)—there is still considerable room for improvement. Key challenges  
 057 include accurately detecting subtle semantic inconsistencies and effectively leveraging these detected  
 058 inconsistencies to refine the formalization.

059 To address these two challenges, we proposed an automated LLM-based formalization framework  
 060 built upon **Logical Concept Decomposition** (LoC-Decomp), which integrates a modular formaliza-  
 061 tion template, an automatic semantic consistency check (ASCC) method, and an iterative re-  
 062 finement method within Lean 4. Specifically, we instruct the LLM to generate Lean 4 code con-  
 063 forming to a predefined template that decomposes the formal statement into multiple declaration  
 064 segments, thereby enabling a semantically complete breakdown. A divide-conquer-merge based  
 065 back-translation process is then applied to more accurately capture subtle semantics in Lean 4. Sub-  
 066 sequently, we perform both segmented and holistic discrepancy detection by prompting the LLM  
 067 to identify potential semantic inconsistencies within individual segments and the entire statement.  
 068 The detected discrepancies are then evaluated against predefined criteria by LLM, and rectification  
 069 suggestions are also provided along with the evaluation procedure. Utilizing these discrepancy de-  
 070 scriptions and recommendations as well as compiler error messages, an iterative refinement strategy  
 071 is implemented to achieve both semantic consistency and syntactic correctness.

072 As in previous works, we conducted extensive experiments on widely used mathematical datasets,  
 073 MATH-500(Lightman et al., 2024) and miniF2F(Zheng et al., 2022) for example, to evaluate the  
 074 effectiveness of our proposed approach. The results indicate that after incorporating the LoC-  
 075 Decomp Lean 4 template and few-shot examples, the single-round (pass@1) formalization suc-  
 076 cess rate on the miniF2F dataset reached 77.25%. With the further integration of semantic con-  
 077 sistency checks, compiler verification, and iterative feedback refinement, the pass rate on miniF2F  
 078 increased to 90.16%.<sup>1</sup> Experiment results and code are available at <https://anonymous.4open.science/r/auto-Formalization-1784>.

079 In summary, the main contributions of this paper are as follows:

- 081 1. We introduced LoC-Decomp, a COT-like Lean 4 template that breaks down the formalization  
 082 process into multiple steps, with theoretical guarantees of expressiveness.
- 083 2. We introduced a novel semantic consistency checking method by decomposing the formaliza-  
 084 tion code. This approach enables LLMs to accurately detect semantic inconsistencies between the  
 085 formalized code and the original problem in complex scenarios.
- 087 3. We presented an iterative feedback-based method for semantic and syntactic correction, allow-  
 088 ing the LLM to leverage identified semantic inconsistencies from the previous step or compiler-  
 089 generated syntax errors to iteratively refine the Lean 4 code.
- 090 4. We conducted extensive experiments on two widely adopted datasets and evaluated the results us-  
 091 ing an automatic evaluation metric supplemented by human assessment, which collectively demon-  
 092 strate the effectiveness of the proposed approach.

## 094 2 BACKGROUND AND RELATED WORK

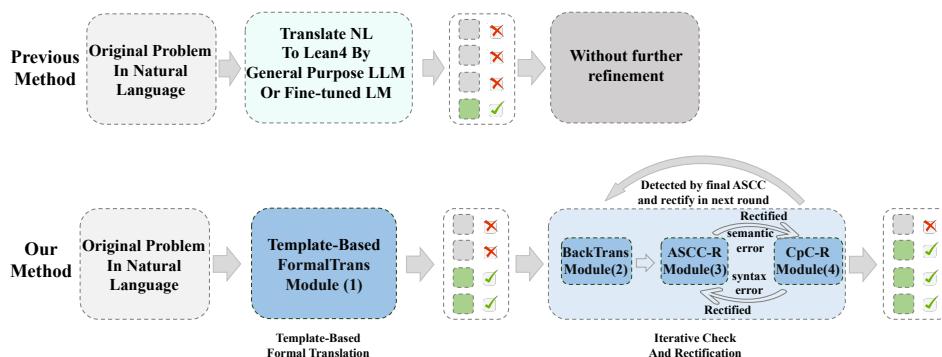
096 **Formal Reasoning:** Recently, a number of studies have emerged that employ formal provers such  
 097 as Lean 4, Coq, and Isabelle to validate the reasoning processes of LLMs (Yang et al., 2023; Wang  
 098 et al., 2024; Li et al., 2024b; Lin et al., 2025a; Alfarano et al., 2024; Huang et al., 2024). As pointed  
 099 out in (Yang et al., 2024), leveraging rigorous formal provers to provide feedback can effectively  
 100 mitigate data scarcity and counteract hallucinations. Automated theorem provers represent one of  
 101 the building blocks of this approach: given a formal proposition, they require the LLM to output  
 102 a proof process, which is then verified using a formal proof system. BFS-prover (Xin et al., 2025)  
 103 through an optimized Best-First Search framework enhanced by expert iteration and policy refine-  
 104 ment. DeepSeek-prover-v2 (Ren et al., 2025) introduces a cold-start reinforcement learning pro-  
 105 cedure that integrates informal mathematical reasoning with formal proof steps through a recursive  
 106 theorem-proving pipeline.

107 <sup>1</sup>All these results are under the ASCC-3-MV metric with DeepSeek-V3 as base model, see section 4 for  
 108 more information.

108 **Autoformalization:** Unlike theorem proving, autoformalization does not generate proofs for theorems; rather, it converts natural language statements into formal specifications (Weng et al., 2025).  
 109 Autoformalization thus acts as a bridge between informal and formal mathematics. Traditional rule-  
 110 based autoformalization methods (Pathak, 2024) are limited by their manual design, fragility to un-  
 111 seen constructs, and poor semantic disambiguation. In contrast, methods based on Large Language  
 112 Models (LLMs) offer greater flexibility and are capable of capturing subtle or rare linguistic pat-  
 113 terns that might be overlooked by human experts during rule design. LLM-based autoformalization  
 114 primarily follows two research directions: fine-tuning on synthetically generated data (Jiang et al.,  
 115 2025; 2023b;a; Azerbayev et al., 2023a) and prompt-based (in-context learning) approaches. Some  
 116 researchers observed that auto-informalization is easier than autoformalization (Wu et al., 2022). This  
 117 observation has motivated subsequent work that employs LLM-based back-translation to verify the  
 118 correctness of formalized outputs (Li et al., 2024a). Another line of research, parallel to our work,  
 119 explores the use of Retrieval-Augmented Generation to improve the formalization abilities of large  
 120 language models (Liu et al., 2025a). A training based evaluation method is proposed by Lu et al.  
 121 (2024a) to detect misalignment in formalization, but this method can only provide a numerical score  
 122 without targeted revision suggestions. For feedback refinement, several studies have utilized com-  
 123 piler error messages to improve the formalization results generated by LLMs(Liu et al., 2025b; Lu  
 124 et al., 2024b; Zhang et al., 2024). The work of KELPS (Zhang et al., 2025) introduced an interme-  
 125 diate language to facilitate concept decomposition; however, it did not integrate this decomposition  
 126 idea into semantic consistency checking. And FIMO(Liu et al., 2023) was the first to introduce a  
 127 mechanism that integrates semantic feedback with syntactic error correction. However, a significant  
 128 limitation is that their semantic feedback mechanism required manual involvement. In summary, an  
 129 automated approach that integrates semantic inconsistency feedback with compiler errors to sim-  
 130 taneously achieve semantic consistency and syntactic correctness remains unexplored.  
 131

### 3 LOC-DECOMP BASED ITERATIVELY REFINEMENT FRAMEWORK

134 As shown in Figure 1, our proposed autoformalization framework consists of four modules: (1)  
 135 template-based formal translation (FormalTrans), (2) decomposition-based back translation (Back-  
 136 Trans), (3) automatic semantic consistency check and iterative rectification (ASCC-R), and (4) com-  
 137 piler check and iterative rectification (CpC-R).  
 138



151 Figure 1: Overview of our Loc-Decomp based iteratively refinement framework (bottom) and an  
 152 overview of previous method (top). In comparison with previous methods that select one from mul-  
 153 tiple generated candidates, our approach introduces iterative feedback and refinement, eliminat-  
 154 ing the need to generate multiple candidates and gains cumulative knowledge from each iteration.  
 155

156 When processing a mathematical problem formalization task, (1) the FormalTrans module first com-  
 157 bines the problem with task requirements, few-shot examples (more details discussion about the few-  
 158 shot examples are available in A.6), and a Lean 4 template to form a complete prompt, which is then  
 159 provided to the LLM for formalization. (2) The formalization result is then send to the BackTrans  
 160 module to get a natural language description of the Lean 4 code. (3,4) An alternating semantic con-  
 161 sistency checking with rectification and compilation error checking with rectification are conducted  
 iteratively by the ASCC-R module and CpC-R module. Inside the ASCC-R module, a submodule

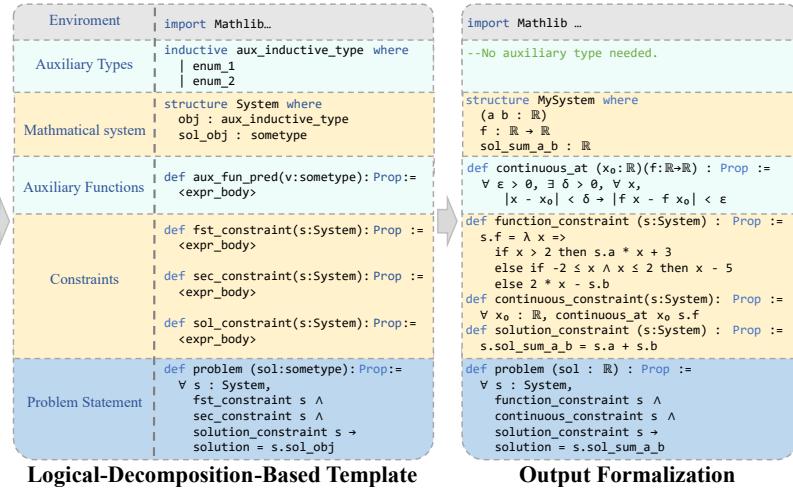
162 named ASCC conduct semantic consistency checking and propose modification suggestions is introduced.  
 163 These suggestions are then addressed through rectification. The check and the rectification  
 164 are iteratively repeated until semantic consistency is achieved (or the maximum iteration count is  
 165 reached in the loop which we call inner semantic loop). Subsequently, if the Lean 4 code pass the  
 166 semantic check, it is sent to the CpC-R module for compiler check and syntax corrections until the  
 167 code passes the compiler check (or the maximum iteration count is reached in the loop which we call  
 168 inner syntax loop). The process of semantic checking and correction, as well as compiler check and  
 169 correction, must be alternatively repeated (which we call the outer loop) until the code consecutively  
 170 passes both semantic and compiler checks. The final formalization result is then output.

171 In Section 3.1, we introduce the formal translation module which contains a template that decomposes  
 172 Lean 4 code into predefined components, thereby support fine-grained back translation, se-  
 173 mantic verification, and targeted rectification in subsequent stages. Section 3.2 then introduces the  
 174 back translation module which adopted a divide-conquer-merge strategy and the semantic consis-  
 175 tency check submodule which depend on the BackTrans module and serve as a component in the  
 176 ASCC-R module. Finally, Section 3.3 concisely describes the compiler check and rectification mod-  
 177 ule and introduces how the ASCC-R and CpC-R work in a iterative way.

### 3.1 FORMAL TRANSLATION MODULE

User Prompt: Formalize the following problem into lean4 and adhering to the template provided.  

$$f(x) = \begin{cases} ax + 3, & \text{if } x > 2 \\ x - 5, & \text{if } -2 \leq x \leq 2 \\ 2x - b, & \text{if } x < -2 \end{cases}$$
  
 Find  $a + b$  if  $f(x)$  is continuous.



198 Figure 2: Formalization template and a concrete example. The left is our proposed Template with  
 199 place holders, and the right is an example for formalizing a problem about the property of a con-  
 200 tinuous piecewise function. **Please note that the example shown in this figure was manually designed**  
 201 **to demonstrate the template structure. For examples generated by large language models (LLMs),**  
 202 **we refer readers to Appendix F.**

203 Unlike previous works that formulate the Lean 4 formalization as a theorem with sorry, as shown in  
 204 Figure 2 we formulate the original statement as a predicate named `problem_statement`, which  
 205 relies on a series of previously declared definitions. This design enables LLMs to organize the Lean  
 206 4 content in a more structured manner, thereby enhancing performance as demonstrated in our  
 207 experiments. The formulation this template<sup>2</sup> based on the following conceptual insight: any mathematical  
 208 problem amounts to an investigation of the properties of some mathematical system (i.e. mathemat-  
 209 ical structure). In our work, we represent such a mathematical system as a `structure` declaration  
 210 in Lean 4 named `MySystem` along with a series of predicates that describing the properties of this  
 211 structure. Such a system comprises multiple mathematical objects, together with a set of con-  
 212 straints that restrict the properties or relationships among these objects, thereby defining the specific  
 213 mathematical structure under study. Such an intuition is kind of based on Discourse Representation

214  
 215 <sup>2</sup>This template applies to both solving and proof-oriented problems. The main text focuses on solving-type  
 problems, but the framework can be adapted to proof-type problems with minor adjustments (see A.5).

Theory, which is introduced to mathematical formalization by Ganesalingam (2013). A detailed explanation for this template is available in A.1.

To make sure the template is fully complied with, a verification method is required. Since this template framework imposes requirements beyond the syntax rules of the Lean 4 compiler, a lightweight parser is implemented to serve as template verifier<sup>3</sup>. If the requirements are not met, it returns corresponding feedback to prompt the LLM to regenerate the code. This process iterates until all conditions are satisfied or the maximum iteration limit is reached. We emphasize that although this template is demonstrated using Lean 4, it can be easily adapted to other theorem provers like Coq or Isabella, a detailed discussion is provided in A.7 and details about the parser and the feedback strategy are available in A.2.

Since such a template imposes additional requirements on top of Lean syntax, the set of all Lean 4 code that satisfies the template forms a proper subset of all Lean 4 code that meets the syntactic requirements. This implies that Lean 4’s expressive power may be constrained under this template, as certain formal declarations might not be convertible into the template’s structure. To provide a theoretical guarantee for the expressiveness of the template, we propose the following theorem3.1 and provide its proof in the A.4.

**Definition 3.1.** LoC-Decomp counterpart: For a complete Lean 4 proposition string(by complete we mean that this string contains a proposition defined by theorem or lemma and all its dependencies), we define its LoC-Decomp counterpart as a string that is semantically equivalent to the Lean 4 proposition and satisfies the LoC-Decomp template requirements. Here, satisfying the LoC-Decomp template requirements means that the string can be accepted by our parser that encodes the syntactic rules of the template.

**Theorem 3.1.** *For any complete Lean 4 proposition string, there exists at least one LoC-Decomp counterpart.*

### 3.2 BACK TRANSLATION MODULE AND ASCC SUBMODULE

With the aid of the aforementioned template, we can now back-translate the Lean 4 code into natural language while preserving its nuanced semantic information through a divide-conquer-merge strategy, as shown in the top half of Figure 3. This back translation serves as auxiliary information to enhance the model’s understanding of the code and to further evaluate its consistency with the original statement. Specifically, we first convert the Lean 4 code back into its natural language equivalent, carefully retaining subtle semantic details. Both the back-translated text and the Lean 4 code along with the original statement are then provided to the LLM to assess their consistency.

Given a Lean 4 code which is a formal version of a mathematical statement, the back translation procedure is to translate the Lean 4 code into natural language while maintaining the semantic essence. For complex mathematical statements, it is often unreliable to use an LLM to translate complete Lean 4 code into natural language in one go, because LLMs struggles to detect the subtle semantic logic implicit in Lean 4 code. To address this difficulty, we adopt an approach that (1) decomposes(divide) the Lean 4 code into several semantically self-contained segments<sup>4</sup>; (2) provide each segment to LLM for translation(conquer), and (3) subsequently integrates them into a cohesive whole based on the logical relationships among the parts(merge). A detailed discussion of the divide and merge steps will be presented in Appendix C, omitting the conquer step due to its relative simplicity.

For the automatic semantic consistency checking submodule, as shown in the bottom half of Figure 3, we employ a four-stage strategy: identifying all potential discrepancies, evaluating the significance of each discrepancy, synthesizing critical discrepancies to determine an overall consistency

<sup>3</sup>Lean4’s expressiveness may be impaired by this mandatory compliance of this template, a detailed discussion is provided in the A.4

<sup>4</sup>The term “semantically self-contained” here refers to segments that encapsulate complete and independent units of meaning, where all necessary semantic components—such as definitions or premises—are explicitly contained within the segment itself.

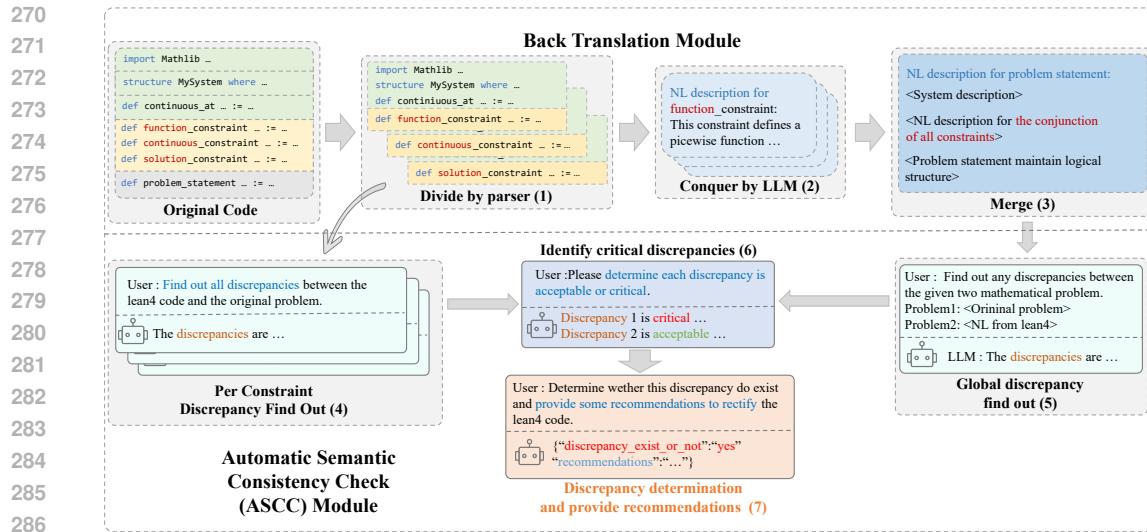


Figure 3: Back translation by divide-conquer-merge strategy and the automatic semantic consistency check.

level, and re-examining each discrepancy in cases not fully consistent—if any discrepancy is confirmed, the output is deemed semantically inconsistent. Specifically, (4) segmented and holistic discrepancy detection are employed to identify differences between the formalization and the problem description. The former targets subtle, localized inconsistencies, whereas the latter focuses on overall, structural issues; (5) Collective evaluation of identified discrepancies to classify them as critical or acceptable based on predefined criteria, followed by determining an overall consistency level according to all critical discrepancies, the consistency level include: Fully consistent, Consistent without loss of generality, Inconsistent; (6) Individual re-evaluation of each discrepancy if the code is not fully consistent; and (7) For each confirmed discrepancy, the LLM must provide correction suggestions, and the formalization is judged not fully consistent. If any of discrepancy exist, the formalization is inconsistent. Detailed explanations of each step and judge criteria as well as consistency level definitions are available in Appendix B and prompt used are provided in Appendix I. The ASCC module enables us to provide targeted discrepancy information and recommendations, allowing the LLM to rectify the Lean 4 code (as described in following section).

### 3.3 JOINT SYN-SEM ITERATIVE RECTIFICATION MODULE

As shown in Figure 4 The iterative rectification procedure is a alternating process of the semantic inconsistency correction and compilation error (i.e., syntax error) correction. Each approach utilizes error feedback to prompt the LLM to reassess its prior output and produce a revised solution. The primary distinction lies in the source of the feedback: semantic error information is derived from semantic consistency checks, while compilation error information is provided by the compiler. In both cases, once error information is obtained, it is appended to the context supplied to the LLM for revision. The updated output then undergoes another round of semantic consistency verification or compilation checking. This iterative cycle continues until the code passes both checks consecutively or the maximum iteration count is reached.

More specifically, our approach involves an iterative process that alternates between semantic and syntactic correction, with the objective of steering the formalization toward semantic consistency and syntactic correctness. For a semantically inconsistent formalization, we first run an inner semantic loop, where semantic corrections are made and re-validated iteratively until success or until reaching the maximum semantic iterations ( $K\text{-sem}$ ). After passing semantic validation, it enters the inner syntactic loop, undergoing compilation checks and syntactic fixes until it compiles successfully or exceeds the allowed syntactic iterations ( $K\text{-syn}$ ). One inner semantic loop followed by one inner syntactic loop forms a Sem–Syn Iter Unit. If the formalization fails to pass both checks consecutively within one unit, it triggers another Sem–Syn Iter Unit; this multi-unit

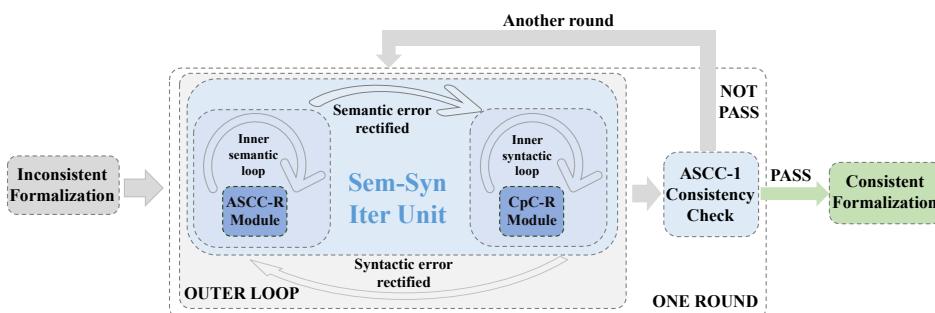


Figure 4: Joint Syn-Sem iterative rectification

process is the OUTER LOOP, which can run up to  $N$  times. After the OUTER LOOP, a final validation for both semantic and syntactic errors is performed—this entire sequence, including at most  $N$  outer loops plus the final check, constitutes one ROUND. Failed cases proceed to the next ROUND, with the entire process repeating for up to  $M$  rounds. A detailed description of this process is provided in Algorithm 1. In this algorithm, the function  $\text{ASCC}(\cdot)$  returns the result of the ASCC check, denoted as  $\text{ASCCLevel}$ , which can take one of three values: Fully Consistent, Consistent Without Loss of Generality, or Inconsistent. Here,  $\text{ASCC-R}(\cdot)$  refers to the ASCC check with rectification, and  $\text{CpC-R}(\cdot)$  denotes the compiler check with rectification.

---

**Algorithm 1** Iterative check and rectification
 

---

**Input:**  $\text{problem, Lean4};$   
**Output:**  $\text{rectifiedLean4, finalASCC};$

```

1:  $\text{ASCCLevel} \leftarrow \text{ASCC}(\text{Lean4}), m \leftarrow 0;$ 
2: while ( $\text{ASCCLevel} = \text{Inconsistent}$  or compiler check not pass) and  $m < M$  do
3:   while ( $\text{ASCCLevel} = \text{null}$  or  $\text{Inconsistent}$ ) and iteration limit not reached do
4:     while  $\text{ASCCLevel} \neq \text{Fullyconsistent}$  and iteration limit not reached do
5:        $\text{Lean4ToNL} \leftarrow \text{Informalization}(\text{Lean4});$ 
6:        $\text{Lean4, ASCCLevel} \leftarrow \text{ASCC-R}(\text{Lean4ToNL}, \text{Lean4}, \text{problem});$ 
7:     end while
8:     if  $\text{ASCCLevel} \neq \text{Inconsistent}$  and compiler check not pass then
9:       while compiler check not pass and iteration limit not reached do
10:         $\text{Lean4} \leftarrow \text{CpC-R}(\text{Lean4});$ 
11:      end while
12:       $\text{ASCCLevel} \leftarrow \text{null};$ 
13:    end if
14:  end while
15:   $\text{rectifiedLean4} \leftarrow \text{Lean4}, m \leftarrow m + 1;$ 
16: end while
17: return  $\text{rectifiedLean4, ASCC}(\text{rectifiedLean4});$ 

```

---

## 4 EVALUATION

### 4.1 EXPERIMENT SETUP

**Datasets:** To evaluate our methods, we employed two widely used public datasets—MATH-500(Lightman et al., 2024) and miniF2F(Zheng et al., 2022) as well as two custom datasets: MATH-ASCC-Eval-150 and MATH-Level5-50. The MATH-500 dataset comprises 500 problems sampled from the MATH-12500(Hendrycks et al., 2021) dataset, covering a variety of problem types and difficulty levels. The miniF2F dataset contains 488 problems sourced from AIME, AMC, and IMO competitions, which is specifically designed for autoformalization by converting solve-type problems into proof-type problems. **To evaluate the performance of our method on more complex mathematical problems, we also adopted PutnamBench(Tsoukalas et al., 2024)**

378 and ProofNet(Azerbayev et al., 2023a) as test datasets for further evaluation. PutnamBench and  
 379 ProofNet contain 522 and 371 undergraduate-level mathematical problems respectively, presenting  
 380 additional challenges for the autoformalization by LLMs.

381 The MATH-ASCC-Eval-150 dataset comprises 150 problems sampled from MATH-500 across dif-  
 382 ficulty levels, each accompanied by Lean 4 formalizations and human annotations (91 positive,  
 383 59 negative). The negative cases predominantly contain subtle errors, making this dataset suitable  
 384 for evaluating automated consistency-checking methods (see Appendix Appendix E for details).  
 385 In contrast, the MATH-Level5-50 subset consists of 50 randomly selected level-5 problems from  
 386 MATH-500, designed to assess the formalization capability on challenging problems.

387 **Models:** Experiments were conducted using three open source LLMs—including DeepSeek-V3,  
 388 KIMI-K2 and Qwen3-235B(Liu et al., 2024; Yang et al., 2025; Kimi Team et al., 2025)—to validate  
 389 the effectiveness of the proposed method across different LLMs.

390 **Baseline:** For a fair comparison with our method, we implemented several baseline methods for  
 391 three tasks. For the semantic consistency checking task, we introduced two baselines: (1) SC-  
 392 Baseline, where the LLM is provided with both the original problem and its corresponding Lean 4  
 393 code then prompted to assess semantic consistency; and (2) SC-Baseline-BackTrans, which addi-  
 394 tionally supplies a back translation of the Lean 4 code. For the pass@1 formalization task, we de-  
 395 veloped (3) Baseline, which employs a basic few-shot prompt to guide the LLM in formalizing the  
 396 given problem. For the iterative formalization task, we designed (4) Baseline-iter, which implements  
 397 an iterative method identical to ours but employs a basic semantic checker like the SC-Baseline. **To**  
 398 **compare our method with the Supervised Fine-Tuning based models, we designed SFT-Baseline by**  
 399 **selecting DeepSeek-Prover-V2(Ren et al., 2025) as the base model, and other configurations are ex-**  
 400 **actly the same with the Formal-Baseline.** The prompts and detailed configurations for these baseline  
 401 methods are provided in the Appendix I.

402 **Metrics:** For the semantic consistency evaluation, we assessed the ASCC method on the MATH-  
 403 ASCC-Eval-150 dataset as well as the PutnamBench-ASCC-Eval-50 dataset using recall, precision,  
 404 and F1-score. For the back translationi task, we assess the metric of translation success rate by  
 405 human evaluation. For the iterative rectification task, the primary evaluation metric was the ASCC-  
 406 3-MV pass rate—determined through majority voting over three rounds of ASCC—supplemented  
 407 by human-evaluated pass rates. In addition to ASCC-3-MV, we also report the ASCC-3-SV pass  
 408 rate—based on single veto over three rounds of ASCC—as a reference metric, due to its high preci-  
 409 sion but low recall.

410 **Implementation:** In multi-round iterative testing, we set the maximum iteration numbers as  
 411  $K_{sem} = 2$ ,  $K_{syn} = 3$ ,  $N = 5$ ,  $M = 3$ . Across all evaluations, the temperature parameter  
 412 was set to 0.7 for ASCC and 0.3 for all other requests, consistently applied to all LLMs. And all the  
 413 baseline methods are conducted with DeepSeek-V3.

## 414 415 4.2 EXPERIMENT RESULTS

416 **ASCC-3 Evaluation.** We first evaluate the alignment between the ASCC and human judgment  
 417 criteria for assessing formalization semantic consistency. As shown in Figure 5, on the MATH-  
 418 ASCC-Eval-150 dataset, the ASCC-3-MV metric achieves a precision of 0.90, a recall of 0.82,  
 419 and a F1-score of 0.86. Compared to the baseline, precision shows significant improvement—a  
 420 key focus, as it reflects the method’s capability to accurately identify errors. Although precision  
 421 slightly decreases relative to ASCC-3-SV, recall remains at a reasonably high level. We therefore  
 422 conclude that ASCC-3-MV aligns most closely with human evaluation criteria. Owing to its strong  
 423 performance in detecting negative instances, we propose ASCC-3-MV as a standard for evaluating  
 424 the performance of our methods, while still providing ASCC-3-SV as a more stringent reference.

425 **Pass@1 with the template and few-shot examples.** As an ablation study, we evaluate our approach  
 426 without the iterative rectification process to assess the contribution of LoC-Decomp template to the  
 427 overall pass rate. The results in Table 1 indicate that even without iterative rectification, our method  
 428 still achieves a relatively high pass rate. We attribute this performance to the chain-of-thought-style  
 429 template and the use of few-shot examples based on classification. This comparison confirms that  
 430 our iterative rectification strategy results in a clear performance improvement, with increases in all  
 431 evaluated items.

		ASCC-3-MV		ASCC-3-SV		SC-Baseline		SC-Baseline-BackTrans	
		Human Label	Predicted Label	Human Label	Predicted Label	Human Label	Predicted Label	Human Label	Predicted Label
	Pos	80	17	Pos	54	43	Pos	96	1
	Neg	9	44	Neg	3	50	Neg	45	8
	Pos	80	17	Pos	54	43	Pos	96	1
	Neg	9	44	Neg	3	50	Neg	45	8
	Pos	80	17	Pos	54	43	Pos	96	1
	Neg	9	44	Neg	3	50	Neg	45	8
	P: 0.90 R: 0.82 F1: 0.86	P: 0.95 R: 0.56 F1: 0.70	P: 0.68 R: 0.99 F1: 0.81	P: 0.70 R: 0.98 F1: 0.82					

Figure 5: Confusion matrix of the ASCC-3-MV, ASCC-3-SV, SC-Baseline and SC-Baseline-BackTrans (baseline with back translation) evaluation result.

Table 1: Results without iterative rectification						
Items	DeepSeek-V3		KIMI-K2		Qwen3-235B	
	MV	SV	MV	SV	MV	SV
MATH-500	63.20	44.80	60.20	43.40	<u>65.40</u>	45.60
miniF2F	77.25	54.30	<u>78.57</u>	60.29	76.90	60.29

**Joint Syn-Sem iterative rectification evaluation.** The results of the iterative rectification method are summarized in Table 2, where MV denotes the ASCC-3-MV pass rate, and SV denotes the ASCC-3-SV pass rate. The best-performing method is underlined. After three iterations, our approach achieved a ASCC-3-MV pass rate of 84.00% on MATH-500 and 90.16% on miniF2F when combined with DeepSeek-V3, representing the highest performance among all three models evaluated. The result with iterative correction demonstrates a significant improvement over pass@1, which confirms the effectiveness of our proposed method<sup>5</sup>.

Table 2: Iterative rectification round-3 results						
Items	DeepSeek-V3		KIMI-K2		Qwen3-235B	
	MV	SV	MV	SV	MV	SV
MATH-500	<u>84.00</u>	61.60	77.80	<u>55.60</u>	75.60	57.80
miniF2F	90.16	67.00	<u>91.60</u>	72.90	80.25	61.07

**Human evaluation.** As a supplement to the ASCC-3-MV and ASCC-3-SV metrics, we performed a human evaluation on the MATH-Level5-50 dataset. For comparison, three baseline methods—as outlined in Section 4.1—were implemented. The results indicate that, using only template guidance and few-shot examples, our approach achieves a formalization pass rate that exceeds the baseline by 18% under human evaluation. After three rounds of iterative refinement, the pass rate further increased to 84%, surpassing the baseline by 30%. Even when compared to models fine-tuned with expert iteration, our method achieves a higher pass rate. The results are summarized in Table 3, where "Ours-no-iter" refers to our LoC-Decomp method without iterative rectification, and "Ours-iter" denotes the version with iterative rectification. A case study illustrating the rectification process is provided in Appendix F.

Table 3: Human evaluation on MATH-Level5-50					
Items	Ours-no-iter	Ours-iter	Baseline	Baseline-iter	SFT-Baseline
pass rate	70.00	84.00	52.00	54.00	66.00

**Evaluation on undergraduate-level problems.** For undergraduate-level mathematical problems, our experimental results indicate that under the ASCC-3-MV evaluation standard, our method still

<sup>5</sup>Additional experimental results about the iterative process are available in Appendix G

486 demonstrates a notably high accuracy rate, further evidencing the universality and generalizability  
 487 of our approach. See Table 4 and Table 5 for the results.  
 488

489 Table 4: **Results without iterative rectification on PutnamBench and ProofNet**

491 Items	492 DeepSeek-V3		493 KIMI-K2	
	494 MV	495 SV	496 MV	497 SV
498 PutnamBench	48.37	31.71	53.66	35.16
499 ProofNet	38.42	27.79	43.05	29.16

500 Table 5: **Iterative rectification round-3 results on PutnamBench and ProofNet**

501 Items	502 DeepSeek-V3		503 KIMI-K2	
	504 MV	505 SV	506 MV	507 SV
508 PutnamBench	81.50	58.33	72.36	48.78
509 ProofNet	67.03	48.50	70.57	52.59

510 **Comparative Analysis.** To compare our method with existing approaches, we selected DeepSeek-  
 511 Prover-V2-671B, Goedel-Formalizor-V2-32B(Lin et al., 2025b), and Kimina-Autoformalizer-7B  
 512 for evaluation. To ensure a fair comparison, we adopted the conventional LLM-as-a-judge eval-  
 513 uation framework. The results are summarized in the Table 6.

514 Table 6: **Performance comparison of different theorem provers**

515 Items	516 Ours	517 DeepSeek-Prover	518 Goedel-Formalizor	519 Kimina-AutoFormalizor
520 MATH-500	96.60	84.60	90.20	63.40
521 mini-F2F	97.54	87.18	93.28	87.23
522 PutnamBench	93.09	75.41	78.66	61.99
523 ProofNet	73.57	83.10	71.39	61.31

524 A notable observation is that our method’s performance on the ProofNet dataset is significantly  
 525 lower than that of DeepSeek-Prover-V2. Analysis reveals that the compilation check pass rate of  
 526 our method on ProofNet is only 74%, considerably lower than the over 95% pass rate achieved on  
 527 other benchmark datasets. Manual inspection indicates that most compilation failures are due to  
 528 unresolved type class instances (e.g., “failed to synthesize” errors). We hypothesize that this issue is  
 529 related to ProofNet’s heavy reliance on advanced Mathlib usage patterns. Since the current method  
 530 does not incorporate a retrieval-augmented generation (RAG) mechanism, the large language model  
 531 struggles to accurately retrieve and incorporate relevant Mathlib definitions, thereby compromising  
 532 compilation success. It should be noted that this limitation is not an inherent flaw of the method  
 533 itself, but rather an orthogonal challenge that can be effectively addressed by integrating RAG-based  
 534 extensions.

535 

## 5 LIMITATION

536 Human evaluation in this study was conducted on a dataset of limited scale, a constraint commonly  
 537 encountered in this field due to the labor-intensive nature of such evaluations. While this is con-  
 538 sistent with the practices of related works that face similar scalability challenges, expanding the  
 539 size of the human-evaluated dataset in future research could further strengthen the reliability and  
 540 generalizability of our proposed approach.

540            **6 THE USE OF LARGE LANGUAGE MODELS**  
 541

542            This study utilized Large Language Models (LLMs) in two distinct roles:  
 543

544            1. Language Polishing and Editing: LLMs were employed as an assistive tool to enhance the lan-  
 545            guage quality and clarity of the manuscript. The initial draft was entirely authored by the researcher,  
 546            after which the LLM provided suggestions to improve sentence fluency, grammar, and academic  
 547            tone. All conceptual contributions, arguments, and factual assertions originated from the author.  
 548            The final manuscript was thoroughly reviewed and approved by the author, who takes full responsi-  
 549            bility for its content.

550            2. LLM as a Research Subject: A key aspect of this research involves the evaluation of LLM  
 551            capabilities. The outputs generated by the model were systematically analyzed as a primary focus  
 552            of the study. The corresponding methodology is elaborated in the main text.

553            **7 REPRODUCIBILITY STATEMENT**  
 555

556            To ensure the reproducibility of our work, we have made our complete codebase, along with the raw  
 557            experimental results and detailed instructions for setting up the environment, publicly available. All  
 558            materials have been submitted to an anonymous repository for blind peer review and will be retained  
 559            upon publication.

560            **REFERENCES**  
 561

563            Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Ale-  
 564            man, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical  
 565            report. *arXiv preprint arXiv:2303.08774*, 2023.

566            Alberto Alfarano, François Charton, and Amaury Hayat. Global lyapunov functions: a long-standing  
 567            open problem in mathematics, with symbolic transformers. *Advances in Neural Information Pro-*  
 568            *cessing Systems*, 37:93643–93670, 2024.

570            Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev,  
 571            and Jeremy Avigad. Proofnet: Autoformalizing and formally proving undergraduate-level math-  
 572            ematics. *CoRR*, abs/2302.12433, 2023a. doi: 10.48550/ARXIV.2302.12433. URL <https://doi.org/10.48550/arXiv.2302.12433>.

574            Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Al-  
 575            bert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model  
 576            for mathematics. *arXiv preprint arXiv:2310.10631*, 2023b.

577            Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art:*  
 578            *The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS  
 579            Series. Springer, 2004. ISBN 978-3-642-05880-6. doi: 10.1007/978-3-662-07964-5. URL <https://doi.org/10.1007/978-3-662-07964-5>.

582            Leonardo Mendonça de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von  
 583            Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middel-  
 584            dorp (eds.), *Automated Deduction - CADE-25 - 25th International Conference on Automated*  
 585            *Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in*  
 586            *Computer Science*, pp. 378–388. Springer, 2015. doi: 10.1007/978-3-319-21401-6\_26. URL  
 587            [https://doi.org/10.1007/978-3-319-21401-6\\_26](https://doi.org/10.1007/978-3-319-21401-6_26).

588            Mohan Ganesalingam. *The Language of Mathematics: A Linguistic and Philosophical Investigation*.  
 589            Lecture Notes in Computer Science. Springer Berlin, Heidelberg, 1 edition, 2013. ISBN 978-  
 590            3-642-37011-3. doi: 10.1007/978-3-642-37012-0. URL <https://doi.org/10.1007/978-3-642-37012-0>.

592            Georges Gonthier. The four colour theorem: Engineering of a formal proof. In *Asian Symposium*  
 593            *on Computer Mathematics*, pp. 333–333. Springer, 2007.

594 Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song,  
 595 and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv*  
 596 *preprint arXiv:2103.03874*, 2021.

597 Yinya Huang, Xiaohan Lin, Zhengying Liu, Qingxing Cao, Huajian Xin, Haiming Wang, Zhenguo  
 598 Li, Linqi Song, and Xiaodan Liang. Mustard: Mastering uniform synthesis of theorem and proof  
 599 data, 2024. URL <https://arxiv.org/abs/2402.08957>.

600 Albert Q Jiang, Wenda Li, and Mateja Jamnik. Multilingual mathematical autoformalization. *arXiv*  
 601 *preprint arXiv:2311.03755*, 2023a.

602 Albert Q. Jiang, Wenda Li, and Mateja Jamnik. Multi-language diversity benefits autoformalization.  
 603 In *Proceedings of the 38th International Conference on Neural Information Processing Systems*,  
 604 NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc. ISBN 9798331314385.

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

609 Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen,  
 610 Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong,  
 611 Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao,  
 612 Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang  
 613 Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu,  
 614 Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin,  
 615 Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao  
 616 Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin  
 617 Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu,  
 618 Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe  
 619 Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo  
 620 Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi,  
 621 Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng  
 622 Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang,  
 623 Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang,  
 624 Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu,  
 625 Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing  
 626 Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie  
 627 Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao,  
 628 Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang  
 629 Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang,  
 630 Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng  
 631 Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou,  
 632 Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence,  
 633 2025. URL <https://arxiv.org/abs/2507.20534>.

634 Zenan Li, Yifan Wu, Zhaoyu Li, Xinming Wei, Xian Zhang, Fan Yang, and Xiaoxing Ma. Autoformalize  
 635 mathematical statements by symbolic equivalence and semantic consistency. In *The Thirty-eighth Annual  
 636 Conference on Neural Information Processing Systems*, 2024a. URL  
 637 <https://openreview.net/forum?id=8ihVBYPMV4>.

638 Zenan Li, Zhi Zhou, Yuan Yao, Xian Zhang, Yu-Feng Li, Chun Cao, Fan Yang, and Xiaoxing Ma.  
 639 Neuro-symbolic data generation for math reasoning. *Advances in Neural Information Processing  
 640 Systems*, 37:23488–23515, 2024b.

641 Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan  
 642 Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth  
 643 International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=v8L0pN6EOi>.

644 Yong Lin, Shange Tang, Bohan Lyu, Jiayun Wu, Hongzhou Lin, Kaiyu Yang, Jia Li, Mengzhou Xia,  
 645 Danqi Chen, Sanjeev Arora, et al. Goedel-prover: A frontier model for open-source automated  
 646 theorem proving. *arXiv preprint arXiv:2502.07640*, 2025a.

648 Yong Lin, Shange Tang, Bohan Lyu, Ziran Yang, Jui-Hui Chung, Haoyu Zhao, Lai Jiang, Yihan  
 649 Geng, Jiawei Ge, Jingruo Sun, Jiayun Wu, Jiri Gesi, Ximing Lu, David Acuna, Kaiyu Yang,  
 650 Hongzhou Lin, Yejin Choi, Danqi Chen, Sanjeev Arora, and Chi Jin. Goedel-prover-v2: Scaling  
 651 formal theorem proving with scaffolded data synthesis and self-correction, 2025b. URL <https://arxiv.org/abs/2508.03613>.

652

653 Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao,  
 654 Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint*  
 655 *arXiv:2412.19437*, 2024.

656

657 Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju,  
 658 Chuanyang Zheng, Yichun Yin, Lin Li, Ming Zhang, and Qun Liu. Fimo: A challenge for-  
 659 mal dataset for automated theorem proving, 2023. URL <https://arxiv.org/abs/2309.04295>.

660

661

662 Qi Liu, Xinhao Zheng, Xudong Lu, Qinxiang Cao, and Junchi Yan. Rethinking and improving  
 663 autoformalization: Towards a faithful metric and a dependency retrieval-based approach. In  
 664 *The Thirteenth International Conference on Learning Representations*, 2025a. URL <https://openreview.net/forum?id=hUb2At2DsQ>.

665

666

667 Xiaoyang Liu, Kangjie Bao, Jiashuo Zhang, Yunqi Liu, Yuntian Liu, Yu Chen, Yang Jiao, and  
 668 Tao Luo. Atlas: Autoformalizing theorems through lifting, augmentation, and synthesis of data,  
 669 2025b. URL <https://arxiv.org/abs/2502.05567>.

670

671 Jianqiao Lu, Yingjia Wan, Yinya Huang, Jing Xiong, Zhengying Liu, and Zhijiang Guo. For-  
 672 malalign: Automated alignment evaluation for autoformalization, 2024a. URL <https://arxiv.org/abs/2410.10135>.

673

674 Jianqiao Lu, Yingjia Wan, Zhengying Liu, Yinya Huang, Jing Xiong, Chengwu Liu, Jianhao Shen,  
 675 Hui Jin, Jipeng Zhang, Haiming Wang, Zhicheng Yang, Jing Tang, and Zhijiang Guo. Process-  
 676 driven autoformalization in lean 4, 2024b. URL <https://arxiv.org/abs/2406.01940>.

677

678 Shashank Pathak. Gflean: An autoformalisation framework for lean via gf, 2024. URL <https://arxiv.org/abs/2404.01234>.

679

680 Lawrence C. Paulson. *Isabelle - A Generic Theorem Prover (with a contribution by T. Nipkow)*,  
 681 volume 828 of *Lecture Notes in Computer Science*. Springer, 1994. ISBN 3-540-58244-4. doi:  
 682 10.1007/BFB0030541. URL <https://doi.org/10.1007/BFb0030541>.

683

684 Z. Z. Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue  
 685 Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, Z. F. Wu, Zhibin Gou, Shirong Ma, Hongxuan Tang,  
 686 Yuxuan Liu, Wenjun Gao, Daya Guo, and Chong Ruan. Deepseek-prover-v2: Advancing for-  
 687 mal mathematical reasoning via reinforcement learning for subgoal decomposition, 2025. URL  
 688 <https://arxiv.org/abs/2504.21801>.

689

690 Christian Szegedy. A promising path towards autoformalization and general artificial intelligence.  
 691 In *International Conference on Intelligent Computer Mathematics*, pp. 3–20. Springer, 2020.

692

693 Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut,  
 694 Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly  
 695 capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

696

697 Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong. Solving olympiad geometry with-  
 698 out human demonstrations. *Nature*, 625(7995):476–482, 2024. ISSN 1476-4687. doi: 10.1038/  
 699 s41586-023-06747-5. URL <https://doi.org/10.1038/s41586-023-06747-5>.

700

701 George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings, Ami-  
 702 tayush Thakur, and Swarat Chaudhuri. Putnambench: Evaluating neural theorem-provers on the  
 703 putnam mathematical competition, 2024. URL <https://arxiv.org/abs/2407.11214>.

702 Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhi-  
 703 fang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In  
 704 Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meet-  
 705 ing of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439,  
 706 Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/  
 707 2024.acl-long.510. URL <https://aclanthology.org/2024.acl-long.510/>.

708 Ke Weng, Lun Du, Sirui Li, Wangyue Lu, Haozhe Sun, Hengyu Liu, and Tiancheng Zhang. Auto-  
 709 formalization in the era of large language models: A survey, 2025. URL <https://arxiv.org/abs/2505.23486>.

710  
 711 Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and Chris-  
 712 tian Szegedy. Autoformalization with large language models. In *Proceedings of the 36th Inter-  
 713 national Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA,  
 714 2022. Curran Associates Inc. ISBN 9781713871088.

715  
 716 Ran Xin, Chenguang Xi, Jie Yang, Feng Chen, Hang Wu, Xia Xiao, Yifan Sun, Shen Zheng, and  
 717 Ming Ding. BFS-prover: Scalable best-first tree search for LLM-based automatic theorem prov-  
 718 ing. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.),  
 719 *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Vol-  
 720 ume 1: Long Papers)*, pp. 32588–32599, Vienna, Austria, July 2025. Association for Compu-  
 721 tational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.1565. URL  
 722 <https://aclanthology.org/2025.acl-long.1565/>.

723  
 724 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang  
 725 Gao, Chengan Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu,  
 726 Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin  
 727 Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang,  
 728 Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui  
 729 Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang  
 730 Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger  
 731 Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan  
 732 Qiu. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.

733  
 734 Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil,  
 735 Ryan Prenger, and Anima Anandkumar. Leandojo: theorem proving with retrieval-augmented  
 736 language models. In *Proceedings of the 37th International Conference on Neural Information  
 737 Processing Systems*, NIPS '23, Red Hook, NY, USA, 2023. Curran Associates Inc.

738  
 739 Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn  
 740 Song. Formal mathematical reasoning: A new frontier in ai, 2024. URL <https://arxiv.org/abs/2412.16075>.

741  
 742 Huaiyuan Ying, Zijian Wu, Yihan Geng, JIayu Wang, Dahua Lin, and Kai Chen. Lean workbook:  
 743 A large-scale lean problem set formalized from natural language math problems. In *The Thirty-  
 744 eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*,  
 745 2024. URL <https://openreview.net/forum?id=Vcw3vzjHDb>.

746  
 747 Jiyao Zhang, Chengli Zhong, Hui Xu, Qige Li, and Yi Zhou. Kelps: A framework for verified multi-  
 748 language autoformalization via semantic-syntactic alignment, 2025. URL <https://arxiv.org/abs/2507.08665>.

749  
 750 Lan Zhang, Xin Quan, and Andre Freitas. Consistent autoformalization for constructing math-  
 751 ematical libraries. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Pro-  
 752 ceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp.  
 753 4020–4033, Miami, Florida, USA, November 2024. Association for Computational Linguistics.  
 doi: 10.18653/v1/2024.emnlp-main.233. URL [https://aclanthology.org/2024.emnlp-main.233/](https://aclanthology.org/2024.emnlp-main.233).

754  
 755 Kunhao Zheng, Jesse Michael Han, and Stanislas Polu. minif2f: a cross-system benchmark for  
 756 formal olympiad-level mathematics. In *International Conference on Learning Representations*,  
 757 2022. URL <https://openreview.net/forum?id=9ZPegFuFTFv>.

756 Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q Weinberger, and Yuhuai Wu.  
757 Don't trust: Verify-grounding llm quantitative reasoning with autoformalization. *arXiv preprint*  
758 *arXiv:2403.18120*, 2024.  
759  
760  
761  
762  
763  
764  
765  
766  
767  
768  
769  
770  
771  
772  
773  
774  
775  
776  
777  
778  
779  
780  
781  
782  
783  
784  
785  
786  
787  
788  
789  
790  
791  
792  
793  
794  
795  
796  
797  
798  
799  
800  
801  
802  
803  
804  
805  
806  
807  
808  
809

810 APPENDIX A DETAILED DISCUSSION ABOUT FORMALIZATION TEMPLATE  
811812 A.1 TEMPLATE DETAILS  
813814 As the main text said, a mathematical problem can be formally represented by declaring a mathemat-  
815 ical system consisting of several objects and a collection of constraints, the objective of a mathemat-  
816 ical problem is to examine a certain property (maybe a solve type problem or a proof type problem,  
817 but both are examining a certain property.) of this system. In our work, the formalization template  
818 consists of six sequential steps (For solve type problems):819 1. Environment Declaration: A fixed, predefined setup is used to import basic Mathlib dependen-  
820 cies, open namespaces, and define notations. This prevents LLM hallucinations and ensures a stable,  
821 sufficient base environment. It is noteworthy that, due to the presence of hallucination, the LLM may  
822 occasionally invoke non-existent Mathlib declarations or misuse existing ones during the generation  
823 of Lean 4 code. This can result in code that is semantically coherent yet fails compilation—where  
824 “semantically coherent” refers to the scenario in which, assuming all Mathlib declarations refer-  
825 enced by the model conform to the semantics intended by the LLM, the overall logic of the Lean  
826 4 code remains correct. Moreover, even when provided with compilation error feedback, the issue  
827 often remains challenging to resolve. To mitigate this problem, two potential solutions are consid-  
828 ered: first, the use of Retrieval-Augmented Generation (RAG) to enhance the LLM’s awareness of  
829 Mathlib declarations; second, reducing reliance on Mathlib by minimizing the use of its declarations  
830 wherever possible. In the dataset we processed—which involves high-school-level mathematics and  
831 requires relatively few advanced Mathlib features—we adopted the latter strategy. While this may  
832 constrain the generalizability of our method to more advanced mathematical domains, it is important  
833 to note that RAG remains compatible with our approach. By integrating more sophisticated RAG  
834 techniques, our method can be extended to tackle higher-level mathematical problems.835  
836  
837  
838 1 import Mathlib  
839 2 --> declare\_environment  
840 3 open Real InnerProductGeometry Matrix Topology Filter ENNReal  
841 Polynomial Classical Complex  
842 4 notation " $\mathbb{R}^n$ " => EuclideanSpace ℝ (Fin n)  
843 5 notation " $M[m, n]$ " => Matrix (Fin m) (Fin n) ℝ  
844 6 notation " $\langle x, y \rangle$ " => @inner ℝ \_ \_ x y  
845 7 variable {V : Type} [NormedAddCommGroup V] [InnerProductSpace ℝ V]  
846 8 noncomputable section  
847 9  
848 10 \-Lean4 Code Here-\  
849 11  
850 12 end851  
852  
853 Figure A-6: The environment defined in the template. All Lean 4 code resides within this environ-  
854 ment, so in what follows, we will simply omit it.  
855856 2. Auxiliary Types Declaration: The LLM declares custom types (e.g., structures, inductive types)  
857 to model complex concepts, leveraging Lean 4’s dependent type system without unnecessary re-  
858 strictions. Because Lean 4’s strong expressiveness stems from its rich dependent type system, in  
859 this step, we allow the LLM to declare any type in any manner, ensuring that the template does not  
860 impose unnecessary restrictions on Lean 4’s type system.861 3. Mathematical System Declaration: A structure is declared to abstractly model the problem, con-  
862 taining all involved mathematical objects (e.g., functions, equations, geometric shapes, groups) and  
863 their types. This step is crucial for abstract modeling of the problem (especially for applied problem  
types). This step requires the LLM to declare a structure that represents the mathematical system of

864 the current problem. By “mathematical system of the problem” we mean a structure that includes  
 865 the naming and type declarations of all mathematical objects involved in the problem. These mathematical  
 866 objects may include equations, functions, sequences, matrices, geometric objects, or more  
 867 abstract mathematical subjects such as groups and topologies.

868 4. Auxiliary Functions/Predicates Declaration: Helper functions or predicates are defined to simplify  
 869 the expression of subsequent logical constraints.

870 5. Constraints Declaration: The LLM declares predicates that define the logical constraints and  
 871 relationships between the objects in the mathematical system. This step is crucial for expressing  
 872 logical constraints. It requires the LLM to establish constraints on the properties or relationships  
 873 among various objects in the mathematical system (through the declaration of predicates).

874 6. Problem Statement Declaration: The overall problem statement is formalized as a propositional  
 875 function over a free variable ‘solution’. The truth value of this propositional function indicates  
 876 whether solution constitutes a valid solution to the original problem, thereby establishing a formal  
 877 correspondence between the Lean 4 representation and the problem’s semantics. Since Lean 4 does  
 878 not have the concept of “solving” we need to transform the original problem of a solving nature into  
 879 a proposition. Unlike previous approaches that declared a theorem and replaced the proof process  
 880 with ‘sorry’, we reformulate the original problem description of the solving type into a propositional  
 881 function. This function requires an input ‘s’, a free variable representing the “solution” The  
 882 semantics of this propositional function are as follows: for any instance of the aforementioned math-  
 883 ematical system, the conjunction of the constraints implies a propositional expression concerning the  
 884 free variable ‘s’.

885  
 886 **A.2 DETAILS ABOUT THE PARSER AND THE RECTIFICATION STRATEGY FOR TEMPLATE  
 887 COMPLIANCE**

888 The parser we have implemented will parse Lean 4 code according to the template. In the auxiliary type declaration section, no additional requirements will be imposed, ensuring the Lean 4 type system remains fully intact. The system declaration section requires that the Lean 4 code must define a structure named `MySystem` using the `structure` keyword. The auxiliary function declaration section mandates that functions must be defined with the `def` keyword, and parameters are strictly prohibited from being of the `MySystem` type (to emphasize the auxiliary nature of these functions). The constraint declaration section requires definitions with the `def` keyword, and parameters must strictly be of the `MySystem` type (highlighting constraints on the system). The problem statement section requires definitions with the `def` keyword and must be an implication expression under universal quantification like ‘ $\forall$  sys : `MySystem`, ...’, where the antecedent consists of all constraint conditions, and the consequent is a predicate term related to the free variable `solution` (or `proof_goal` for proof-type problems).

889 If the generated Lean 4 code does not conform to the template, a rectification mechanism is activated.  
 890 This mechanism provides error feedback from the parser and prompts the LLM to revise the Lean  
 891 4 code by appending the error information to the context. Whenever new Lean 4 code is generated,  
 892 it undergoes parsing and rectification—regardless of the current stage, whether it is formalization,  
 893 semantic rectification, or syntactic rectification. Some parser error information are as follows:

- 900 • Error occurred, constraints must be declared by using `def`.
- 901 • Error: in the step of `declare_mathematical_system` you can only declare one struc-  
 902 ture named `MySystem`.
- 903 • Error: in the step of `declare_the_problem_statement` you can only declare one  
 904 predicate named `problem_statement`.
- 905 • Error, the antecedents in the problem statement’s implication must contain all the con-  
 906 straints declared in the constraint declaration section.
- 907 • ...

908  
 909 **A.3 EXPLANATION FOR NONCOMPUTABLE SECTION**

910 Within the domain of high school mathematics, natural language mathematical descriptions (i.e.,  
 911 classical mathematics) are typically grounded in first-order predicate logic—a framework that differs

918 significantly from the dependent type theory underpinning Lean 4. A key practical distinction lies  
 919 in the treatment of function domains: classical mathematical definitions often implicitly specify the  
 920 domain through contextual cues and the expression of the function itself, whereas Lean 4’s type  
 921 system requires explicit and precise domain declarations.

922 This difference considerably complicates the formalization of classical mathematical texts. Since  
 923 the domain is frequently left implicit in the original discourse, translating such content into Lean 4  
 924 requires inferring the domain—a process that can be non-trivial for more complex functions and is  
 925 sometimes the explicit goal of certain mathematical exercises. However, this act of deduction may  
 926 introduce semantic discrepancies if not carefully aligned with the original intent.

927 In the present work (as opposed to prior approaches), we employ a strategy commonly adopted in the  
 928 formalization of classical mathematics: placing relevant declarations within a noncomputable sec-  
 929 tion. This allows us to avoid prematurely imposing constructive or computational constraints—such  
 930 as enforcing specific function domains—unless they are directly stated in the original text. The  
 931 approach maintains mathematical rigor, as domain constraints remain logically inherent in the defi-  
 932 nitions and can be formally derived and verified during the formalization process.

933 It is important to emphasize that this method does not circumvent the core challenges of formaliza-  
 934 tion; rather, by utilizing feedback from Lean 4’s type system (provided to the LLM during interac-  
 935 tion), we can iteratively and efficiently render these implicit elements explicit while respecting the  
 936 original mathematical meaning.

#### 938 A.4 PROOF FOR THEOREM 3.1

940 Here, we complete the proof of Theorem 3.1 by providing a method to convert any complete Lean 4  
 941 proposition into a LoC-Decomp template and demonstrating that the result generated by this method  
 942 is equivalent to the original Lean 4 proposition.

943 Without loss of generality, the assumption here is that the proposition to be converted is defined by  
 944 `theorem` and named as `theorem_to_convert`, along with a series of dependency declarations  
 945 existing in the context. And the template we selected here is the proof type template. The conver-  
 946 sion process involves placing all the aforementioned dependency declarations into the auxiliary type  
 947 section. In the mathematical system declaration section, the `MySystem` structure contains only one  
 948 object: `proof_goal`. We then need to transform the `theorem_to_convert` into a proposition  
 949 and name it as `theoremConverted_prop`, which can be done manually or simply call `#check`  
 950 `theorem_to_convert` to get the corresponding type that is exactly the proposition of the theo-  
 951 rem. The constraint declaration section includes only one constraint, `proof_goal_constraint`,  
 952 which states that `sys.proof_goal = theoremConverted_prop`. In the problem state-  
 953 ment declaration section, the implication premise contains solely `proof_goal_constraint`,  
 954 and the implication conclusion is `sys.proof_goal`.

955 Once such a conversion is performed, the resulting output will conform to our template and can  
 956 be accepted by the designated parser. Then we have to prove that the `problem_statement` is  
 957 logically equivalent to the `theoremConverted_prop`, and we prove this by Lean 4 itself. For  
 958 the sake of universality, we adopted a generic sorry placeholder to express a proposition. Since our  
 959 proof operates at the meta level, this proposition can be replaced with any proposition, and our proof  
 960 can pass the verification of the Lean 4 compiler. See code 1 for detailed information.

961  
 962  
 963  
 964  
 965  
 966  
 967  
 968  
 969  
 970  
 971

```

972     Code 1: Convert a theorem to it's Loc-Decomp counterpart and prove the equivalence
973
974     -->>declare_auxiliary_types
975     --All the dependent declarations should be declared here.
976
977     -->>declare_mathematical_system
978     structure MySystem where
979         proof_goal : Prop
980
981     -->>declare_auxiliary_functions_or_predicates
982     def theoremConverted_prop : Prop := sorry
983     --The sorry place holder can be replaced by any proposition.
984
985     -->>declare_constraints
986     def proof_goal_constraint (sys:MySystem) : Prop :=
987         sys.proof_goal = theoremConverted_prop
988
989     -->>declare_the_problem_statement
990     def problem_statement : Prop :=
991         ∀ sys : MySystem,
992             proof_goal_constraint sys →
993             sys.proof_goal
994
995     theorem equivalence : problem_statement ↔ theoremConverted_prop := by
996         constructor
997
998         intro h
999         unfold problem_statement at h
1000        let sys : MySystem := { theoremConverted_prop }
1001        have constraint : proof_goal_constraint sys := by
1002            unfold proof_goal_constraint
1003            rfl
1004            have h_sys_proof_goal : sys.proof_goal := h sys constraint
1005            exact h_sys_proof_goal
1006
1007            intro h sys h_constraint
1008            rw [h_constraint]
1009            exact h

```

1004  
1005 Although this conversion is not elegant, it theoretically ensures that our template does not impair  
1006 Lean 4's expressive power in any way.

#### 1008 A.5 MINOR MODIFICATION FOR PROOF TYPE PROBLEMS

1009 When handling proof-type problems, the `sol_object` is replaced by a `proof_goal` of type  
1010 `Prop`, which corresponds to the proof objective in the original problem. Similarly, the `solution-`  
1011 `_constraint` is replaced by `proof_goal_constraint`, which defines the proof goal  
1012 concretely. The free variable `solution` is removed from the `problem_statement`, as proof-type  
1013 problems do not require any free variables. In this context, the consequence in the problem statement  
1014 corresponds to `sys.proof_goal` for all problems. With these minor adjustments, the template  
1015 becomes suitable for proof type problems, and our parser remains effective in detecting any  
1016 violations of the template. The framework only requires a preliminary check of the problem type to  
1017 determine which template to use. See figure 6 for an example.

1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

```

1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

```

```

1 -->declare_auxiliary_types
2 --No auxiliary types needed for this problem.
3
4 -->declare_mathematical_system
5 structure MySystem where
6   (a b : R)
7   f : R → R
8   sol_sum_a_b : R
9
10 -->declare_auxiliary_functions_or_predicates
11 def continuous_at_point (x₀ : R) (f : R → R) : Prop :=
12   ∀ ε > 0, ∃ δ > 0, ∀ x, |x - x₀| < δ → |f x - f x₀| < ε
13
14 -->declare_constraints
15 def function_expression_constraint (sys:MySystem) : Prop :=
16   sys.f = λ x =>
17     if x > 2 then sys.a * x + 3
18     else if -2 ≤ x ∧ x ≤ 2 then x - 5
19     else 2 * x - sys.b
20
21 def global_continuous_constraint (sys:MySystem) : Prop :=
22   ∀ x₀ : R, continuous_at_point x₀ sys.f
23
24 def solution_constraint (sys:MySystem) : Prop :=
25   sys.sol_sum_a_b = sys.a + sys.b
26
27 -->declare_the_problem_statement
28 def problem_statement (solution : R) : Prop :=
29   ∀ sys : MySystem,
30     function_expression_constraint sys ∧
31     global_continuous_constraint sys ∧
32     solution_constraint sys →
33     solution = sys.sol_sum_a_b

```

```

1 -->declare_auxiliary_types
2 --No auxiliary types needed for this problem.
3
4 -->declare_mathematical_system
5 structure MySystem where
6   (a b : R)
7   f : R → R
8   proof_goal : Prop
9
10 -->declare_auxiliary_functions_or_predicates
11 def continuous_at_point (x₀ : R) (f : R → R) : Prop :=
12   ∀ ε > 0, ∃ δ > 0, ∀ x, |x - x₀| < δ → |f x - f x₀| < ε
13
14 -->declare_constraints
15 def function_expression_constraint (sys:MySystem) : Prop :=
16   sys.f = λ x =>
17     if x > 2 then sys.a * x + 3
18     else if -2 ≤ x ∧ x ≤ 2 then x - 5
19     else 2 * x - sys.b
20
21 def global_continuous_constraint (sys:MySystem) : Prop :=
22   ∀ x₀ : R, continuous_at_point x₀ sys.f
23
24 def proof_goal_constraint (sys:MySystem) : Prop :=
25   sys.proof_goal = (sys.a + sys.b = 0)
26
27 -->declare_the_problem_statement
28 def problem_statement : Prop :=
29   ∀ sys : MySystem,
30     function_expression_constraint sys ∧
31     global_continuous_constraint sys ∧
32     proof_goal_constraint sys →
33     sys.proof_goal

```

Figure A-7: Converting the example formalization into proof type

```

1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079

```

```

(* declare_auxiliary_types *)
(* No auxiliary types needed for this problem. *)
(* declare_mathematical_system *)
record MySystem =
  a :: real
  b :: real
  f : real → real
  sol_sum_a_b : real

(* declare_auxiliary_functions_or_predicates *)
definition continuous_at_point :: "real ⇒ (real ⇒ real) ⇒ bool" where
  "continuous_at_point x₀ f ⟷
  (∀ε>0. ∃δ>0. ∀x. |x - x₀| < δ → |f x - f x₀| < ε)"

(* declare_constraints *)
definition function_expression_constraint :: "MySystem ⇒ bool" where
  "function_expression_constraint sys ⟷
  (f sys) = (λx. if x > 2 then (sys.a) * x + 3
  else if -2 ≤ x ∧ x ≤ 2 then x - 5
  else 2 * x - (b sys))"

definition global_continuous_constraint :: "MySystem ⇒ bool" where
  "global_continuous_constraint sys ⟷
  (∀x₀::real. continuous_at_point x₀ (f sys))"

definition solution_constraint :: "MySystem ⇒ bool" where
  "solution_constraint sys ⟷
  sol_sum_a_b sys = sys.a + sys.b"

(* declare_the_problem_statement *)
definition problem_statement :: "real ⇒ bool" where
  "problem_statement solution ⟷
  (problem_statement solution ∧
  (problem_statement solution ∧
  global_continuous_constraint sys ∧
  solution_constraint sys →
  solution = sol_sum_a_b sys))"

```

```

(* declare_auxiliary_types *)
(* No auxiliary types needed for this problem. *)
(* declare_mathematical_system *)
Record MySystem : Type :=
  a : R;
  b : R;
  f : R → R;
  sol_sum_a_b : R
).

(* declare_auxiliary_functions_or_predicates *)
Definition continuous_at_point (x₀ : R) (f : R → R) : Prop :=
  forall ε, ε > 0 →
  exists δ, δ > 0 /\
  forall x, Rabs (x - x₀) < δ → Rabs (f x - f x₀) < ε.

(* declare_constraints *)
Definition function_expression_constraint (sys : MySystem) : Prop :=
  forall x,
  sys.f x =
  if Rgt_dec x 2 then (sys.a) * x + 3
  else if and_dec (Rge_dec x (-2)) (Rle_dec x 2) then (x - 5)
  else (2 * x - sys.b).

Definition global_continuous_constraint (sys : MySystem) : Prop :=
  forall x₀ : R, continuous_at_point x₀ sys.f.

Definition solution_constraint (sys : MySystem) : Prop :=
  sys.sol_sum_a_b = sys.a + sys.b.

(* declare_the_problem_statement *)
Definition problem_statement (solution : R) : Prop :=
  forall sys : MySystem,
  function_expression_constraint sys ∧
  global_continuous_constraint sys ∧
  solution_constraint sys →
  solution = sys.sol_sum_a_b.

```

Figure A-8: Template suitable for Isabella and Coq example

1080

## A.6 DISCUSSION ON FEW-SHOT EXAMPLES

1081

1082

In this work, we adopt a few-shot in-context learning approach with a set of manually curated examples. Our collection comprises 38 Lean 4 codes that conform to the formalization template and cover a variety of problem domains—including functions, probability, combinatorics, and geometry. These examples are drawn from the MATH-12500 dataset (Hendrycks et al., 2021), with only blank overlapping with the MATH-500 dataset. For each problem, we first determine its domain and then retrieve approximately 8 to 16 relevant examples from that domain to use as few-shot demonstrations. In our baseline methods, the same set of few-shot examples is used, with the only modification being that the Lean 4 code is transformed to a theorem with sorry.

1089

1090

1091

## A.7 EXAMPLES FOR ISABELLA AND COQ

1092

1093

1094

1095

1096

Our proposed framework is not only compatible with Lean 4, but can also be adapted to other theorem provers such as Isabelle and Coq—given the provision of a suitable parser and minor prompt modifications. This flexibility stems from the template-based design of our framework, which can be applied to any proof assistant that supports a type system, the definition of structures (such as ‘record’ in Coq), as well as predicates and functions.

1097

1098

1099

1100

1101

For Coq, which—like Lean 4—supports dependent type theory, the adaptation process is relatively straightforward. In the case of Isabelle, which uses a simple type system, certain expressive features may be limited; however, these remain sufficient for handling high-school-level mathematical problems. We illustrate the adaptation of our template to both Isabelle and Coq using a simple example: the same piecewise function problem discussed in the main text.

1102

1103

1104

1105

1106

1107

1108

1109

1110

1111

1112

1113

1114

1115

1116

1117

1118

1119

1120

1121

1122

1123

1124

1125

1126

1127

1128

1129

1130

1131

1132

1133

1134 APPENDIX B ASCC WORKFLOW DETAILS  
11351136 The procedure consists of the following steps:  
11371138 First, we supply the LLM with segmented segments of the Lean 4 code along with their corre-  
1139 sponding natural language translations. The model is then prompted to identify any discrepancies  
1140 between auxiliary functions or constraints and the original problem, under the assumption that all  
1141 other components are correct.1142 Second, the complete Lean 4 code and its full natural language description are provided, and the  
1143 LLM is instructed to detect any potential inconsistencies between the two.1144 Third, all identified discrepancies are compiled and presented to the LLM, along with the original  
1145 problem and the Lean 4 code. The model is then asked to assess whether each discrepancy is critical  
1146 or acceptable based on predefined criteria.1147 Fourth, the LLM is required to evaluate the overall consistency level by considering all critical  
1148 discrepancies that have been identified.  
11491150 Finally, if the code is deemed not fully consistent, each individual discrepancy is isolated and reeval-  
1151 uated by the LLM to confirm its validity. For each confirmed discrepancy, the model must provide  
1152 specific recommendations to amend the Lean 4 code. The formalization is considered inconsistent  
1153 if any discrepancy is judged to be genuine.1154 B.1 ASCC JUDGE CRITERIA  
11551156 **Criterion 1: Object Omission**1157 Not all mathematical objects in problem original are reflected in the Lean 4 code or  
1158 their types mismatch. As long as the objects are correctly reflected, some redundancy  
1159 in the Lean 4 code are acceptable.1160 **Criterion 2: Semantic Alteration**1161 The definitions, properties, or relationships of any mathematical object from problem  
1162 original are not exactly preserved in the Lean 4 code. Some redundancy is permitted  
1163 as long as the core semantics are constrained correctly.1164 **Criterion 3: Constraint Incompleteness**1165 The constraints expressed in problem formal fail to comprehensively represent all ex-  
1166 plicit and implicit constraints present in problem original.1167 **Criterion 4: Over Simplification**1168 The Lean 4 code conducts concrete computation or derivation that simplifies the orig-  
1169 inal problem, leading to a semantic inconsistency.1171 B.2 ASCC CONSISTENCY LEVEL DEFINITIONS  
11721173 **Consistency levels 1: Fully consistent**

1174 Fully consistent by the final criterion.

1175 **Consistency levels 2: Consistent without loss of generality**1176 Consistent without loss of generality: The Lean 4 code formalizes the prob-  
1177 lem by analyzing representative cases. In each of these cases, the final crite-  
1178 rion are fully satisfied, and the general case follows through straightforward  
1179 deduction. Or the formalization rely on equivalent conversions, such as trans-  
1180 forming canonical equations into general form. In such cases, the formula-  
1181 tion can be regarded as consistent without loss of generality.1182 **Consistency levels 3: Inconsistent**1183 Any criterion broken can lead to this, as long as it is not Consistent without  
1184 loss of generality.1185  
1186  
1187

1188 APPENDIX C DIVIDE AND MERGE DETAILS IN BACK TRANSLATION  
1189

1190 **Divide:** We adopted a hierarchical approach to decompose the Lean 4 code into multiple  
1191 self-contained segments. The system segment includes all auxiliary types and the `MySystem`  
1192 structure; the auxiliary function segments comprise each auxiliary function along with its de-  
1193 pendencies and the system segment; the constraint segments encapsulate each constraint together  
1194 with its related auxiliary functions and the system segment. The problem statement itself is not pro-  
1195 cessed separately, as the semantic meaning of the problem statement can be fully constructed from  
1196 the segments described above. The divide strategy simplifies the complex back translation task into  
1197 several simpler subtasks. Ensures that each segment can be accurately translated in isolation with-  
1198 out relying on outer contextual information from other parts of the code, thereby reducing ambiguity  
1199 and errors during the LLM processing phase.

1200 **Merge:** After translating all segments, the natural language description of the Lean 4 code could, in  
1201 principle, be obtained by simply concatenating them and adding a statement of their logical rela-  
1202 tionships. However, such a direct approach would result in a tediously long output, which could distract  
1203 the LLM during the subsequent semantic check step. Instead, we perform a conjunctive combination  
1204 of the constraints in natural language. This is achieved by sequentially merging pairs of constraints  
1205 and instructing the LLM to restate them cohesively, leveraging the fact that their logical relationship  
1206 in the original problem statement is simply a conjunction. The final translation of the Lean 4 code  
1207 consists of three core components: a system description, a auxiliary functions description, and a  
1208 constraints description, which are integrated with a description of their logical relationships.

1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241

1242 APPENDIX D HUMAN EVALUATION CRITERIA FOR SEMANTIC CONSISTENCY  
12431244 The detailed criteria for manual inspection are as follows:  
1245

1246 1, Whether the formalized problem discusses the same mathematical objects as the original problem  
1247 (redundant auxiliary objects are allowed, and individual objects may be expressed as sets);  
1248 2, Whether the logical constraints of the original problem are correctly expressed (under the current  
1249 constraints, all object properties and relationships are consistent with the original problem); if certain  
1250 constraint expressions are not generally correct but are correct in the current context, they are still  
1251 considered valid;  
1252 3, Whether the truth value of the formalized problem statement corresponds exactly to that of the  
1253 original problem—that is, for a given set of concrete objects (including a solution in the case of a  
1254 solve-type problem), the formal statement holds true if and only if the original problem is true;  
1255 4, We impose no additional requirements on the form of the solution (e.g., some problems require  
1256 solution in degrees, but solution in radians are also considered correct as long as they are practically  
1257 equivalent). This is because Lean 4 has limited support for symbolic computation, making it difficult  
1258 to satisfy certain formal requirements (e.g., requiring expressions to be in simplest form).  
1259 5, Consistent without loss of generality: The Lean 4 code formalizes the problem by analyzing rep-  
1260 resentative cases. In each such case, the final criterion is fully satisfied, and the general case follows  
1261 through straightforward deduction. Alternatively, the formalization may rely on equivalent conver-  
1262 sions—such as transforming canonical equations into general form—or the expression may hold  
1263 only in the current specific context, even if not generally valid. In these situations, the formulation  
1264 can be regarded as consistent without loss of generality.  
1265

1266  
1267  
1268 The Lean4 code only  
1269 expresses the horizontal  
1270 asymptote in the positive  
1271 infinity direction and  
1272 overlooks the horizontal  
1273 asymptote at negative infinity.  
1274 However, in this problem, the  
1275 horizontal asymptotes at  
1276 negative infinity and positive  
1277 infinity are the same, so we  
1278 consider this to be a  
1279 consistent representation  
1280 without loss of generality.  
1281

1282 **Problem:**  
1283

1284 The graph of  $f(x) = \frac{2x}{x^2-5x-14}$   
1285 has vertical asymptotes  $x = a$   
1286 and  $x = b$ , and horizontal  
1287 asymptote  $y = c$ . Find  $a +$   
1288  $b + c$ .  
1289

```
1 -->>declare_auxiliary_types
2 -- No auxiliary types needed for this problem
3 -->>declare_mathematical_system
4 structure MySystem where
5   f : ℝ → ℝ
6   a : ℝ
7   b : ℝ
8   c : ℝ
9   sol_sum : ℝ
10 -->>declare_auxiliary_functions_or_predicates
11 def has_vertical_asymptote_at (f : ℝ → ℝ) (x₀ : ℝ) : Prop :=
12   ∃ (L : ℝ), ∃ (ε : ℝ), ε > 0 ∧
13   (∀ (x : ℝ), x - x₀ ∧ abs (x - x₀) < ε → abs (f x) > L)
14 def has_horizontal_asymptote_at (f : ℝ → ℝ) (y₀ : ℝ) : Prop :=
15   ∃ (ε : ℝ), ε > 0 → ∃ (M : ℝ), ∀ (x : ℝ), x > M → abs (f x - y₀) < ε
16 -->>declare_constraints
17 def function_expression_constraint (sys : MySystem) : Prop :=
18   sys.f = λ x => (2 * x) / (x^2 - 5 * x - 14)
19 def vertical_asymptotes_constraint (sys : MySystem) : Prop :=
20   has_vertical_asymptote_at sys.f sys.a ∧
21   has_vertical_asymptote_at sys.f sys.b ∧
22   sys.a - sys.b
23 def horizontal_asymptote_constraint (sys : MySystem) : Prop :=
24   has_horizontal_asymptote_at sys.f sys.c
25 def solution_constraint (sys : MySystem) : Prop :=
26   sys.sol_sum = sys.a + sys.b + sys.c
27 -->>declare_the_problem_statement
28 def problem_statement (solution : ℝ) : Prop :=
29   ∃ sys : MySystem,
30   function_expression_constraint sys ∧
31   vertical_asymptotes_constraint sys ∧
32   horizontal_asymptote_constraint sys ∧
33   solution_constraint sys →
34   solution = sys.sol_sum
```

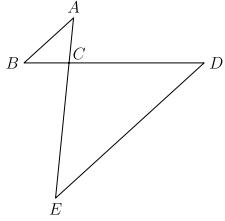
1292 Figure D-9: Case study for consistency without loss of generality.  
1293  
1294  
1295

1296 APPENDIX E MATH-ASCC-EVAL-150 DATASET EXPLANATION IN DETAIL  
12971298 We construct the MATH-ASCC-EVAL-150 dataset through the following steps:  
1299

- 1300 1. Using the method proposed in Section3.1 with DeepSeek-V3, we perform a *pass@1*  
1301 NL2Lean4 formalization procedure. The generated Lean 4 code is then evaluated by  
1302 ASCC-1, and each instance is categorized based on (*level*, ASCC-1 result) pairs.
- 1303 2. From the above outcomes, we randomly sample the following from each category:  
1304
  - 1305 • 10 instances of (level 1, ASCC-1 pass)
  - 1306 • 10 instances of (level 1, ASCC-1 not pass)
  - 1307 • 10 instances of (level 2, ASCC-1 pass)
  - 1308 • 10 instances of (level 2, ASCC-1 not pass)
  - 1309 • 10 instances of (level 3, ASCC-1 pass)
  - 1310 • 10 instances of (level 3, ASCC-1 not pass)
  - 1311 • 20 instances of (level 4, ASCC-1 pass)
  - 1312 • 20 instances of (level 4, ASCC-1 not pass)
  - 1313 • 25 instances of (level 5, ASCC-1 pass)
  - 1314 • 25 instances of (level 5, ASCC-1 not pass)
- 1315 3. We conduct a human evaluation based on predefined criteria to determine the ground-truth  
1316 consistency between each problem and its corresponding Lean 4 code.
- 1317 4. After human evaluation, the final dataset consists of **97 positive cases** and **53 negative**  
1318 **cases**.

1320 In MATH-ASCC-Eval-150, the negative cases contain subtle semantic inconsistency, which are hard  
1321 to detect. Some cases are as follows:1322 **Case studies for nuanced semantic inconsistencies in MATH-ASCC-Eval-150:**  
13231324  
1325  
1326  
1327  
1328  
1329  
1330  
1331  
1332  
1333  
1334  
1335  
1336  
1337  
1338  
1339  
1340  
1341  
1342  
1343  
1344  
1345  
1346  
1347  
1348  
1349

1350  
 1351  
 1352 In the following problem, the  
 1353 Lean4 code on the right fails to  
 1354 capture the fact that points B, C,  
 1355 and D are collinear and points A,  
 1356 C, and F are collinear. This  
 1357 information is implicit in the  
 1358 original problem, resulting in  
 1359 incomplete constraints in the Lean4  
 1360 and inconsistency with the original  
 1361 problem. ASCC-3-MV successfully  
 1362 captured this semantic  
 1363 discrepancy.  
 1364  
 1365  
 1366  
 1367  
 1368 **Problem:**  
 1369 The area of  
 1370  $\triangle ABC$  is 6 square centimeters.  
 1371  $\overline{AB} \parallel \overline{DE}$ .  $BD = 4BC$ . What is the  
 1372 number of square centimeters in the  
 1373 area of  $\triangle CDE$ ?  
 1374  
 1375  
 1376



```

1  -->>declare_auxiliary_types
2  structure Triangle where
3  A : ℝ^2
4  B : ℝ^2
5  C : ℝ^2
6  -->>declare_mathematical_system
7  structure MySystem where
8  ABC : Triangle
9  D : ℝ^2
10 E : ℝ^2
11 sol_area_CDE : ℝ
12 -->>declare_auxiliary_functions_or_predicates
13 def cross_product (v w : ℝ^2) : ℝ :=
14 v.0 * w.1 - v.1 * w.0
15 def triangle_area (t : Triangle) : ℝ :=
16 let AB := t.B - t.A
17 let AC := t.C - t.A
18 abs (cross_product AB AC) / 2
19 def parallel (v w : ℝ^2) : Prop := cross_product v w = 0
20 -->>declare_constraints
21 def area_constraint (sys : MySystem) : Prop := triangle_area sys.ABC = 6
22 def parallel_constraint (sys : MySystem) : Prop :=
23 let AB := sys.ABC.B - sys.ABC.A
24 let DE := sys.E - sys.D
25 parallel AB DE
26 def BD_ratio_constraint (sys : MySystem) : Prop :=
27 let BC := sys.ABC.C - sys.ABC.B
28 let BD := sys.D - sys.ABC.B
29 |BD| = 4 * |BC|
30 def solution_constraint (sys : MySystem) : Prop :=
31 let CDE : Triangle := (sys.ABC.C, sys.D, sys.E)
32 sys.sol_area_CDE = triangle_area CDE
33 -->>declare_the_problem_statement
34 def problem_statement (solution : ℝ) : Prop :=
35 ∀ sys : MySystem,
36  area_constraint sys ∧
37  parallel_constraint sys ∧
38  BD_ratio_constraint sys ∧
39  solution_constraint sys →
40  solution = sys.sol_area_CDE

```

Figure E-10: Case study 1 to illustrate the nuanced inconsistencies in MATH-ASCC-Eval-150

1377  
 1378  
 1379  
 1380 The constraint in the red box  
 1381 does not require that all x  
 1382 satisfying the condition are in  
 1383 the possible\_x set, but only  
 1384 mandates that elements in  
 1385 possible\_x must satisfy the  
 1386 condition. This is inconsistent  
 1387 with the semantics of the  
 1388 original problem. ASCC-3-MV  
 1389 successfully captured this  
 1390 semantic discrepancy.  
 1391  
 1392  
 1393  
 1394 **Problem:**  
 1395 If  $x^2 - x - 1 = 0$ , what  
 1396 is the value of  
 1397  $x^3 - 2x + 1$ ?

```

1  -->>declare_auxiliary_types
2  -- No auxiliary types needed for this problem
3
4  -->>declare_mathematical_system
5  structure MySystem where
6  possible_x : Set ℝ
7  sol_values : Set ℝ
8
9  -->>declare_auxiliary_functions_or_predicates
10 def cubic_expression (x : ℝ) : ℝ := x^3 - 2*x + 1
11
12 -->>declare_constraints
13 def quadratic_equation_constraint (sys : MySystem) : Prop :=
14 ∀ x ∈ sys.possible_x, x^2 - x - 1 = 0
15
16 def solution_constraint (sys : MySystem) : Prop :=
17  sys.sol_values =
18  {y | ∃ x ∈ sys.possible_x, y = cubic_expression x}
19
20 -->>declare_the_problem_statement
21 def problem_statement (solution : Set ℝ) : Prop :=
22  ∀ sys : MySystem,
23  quadratic_equation_constraint sys ∧
24  solution_constraint sys →
25  solution = sys.sol_values

```

Figure E-11: Case study 2 to illustrate the nuanced inconsistencies in MATH-ASCC-Eval-150

1404 APPENDIX F CASE STUDIES FOR ITERATIVE RECTIFICATION ON MATH-50  
1405

1406 As illustrated in Figure F-13, the top half of the diagram demonstrates the output of our method  
1407 during the iterative rectification process. The left portion shows the generated result prior to rec-  
1408 tification, which is inconsistent with the original problem. This inconsistency arises because the  
1409 triangle constraint does not prohibit the points from being collinear. As illustrated in the right por-  
1410 tion of the figure, the ASCC successfully detected and corrected this discrepancy (among others)  
1411 in subsequent iterations, whereas the baseline method made the same error without detecting it. The  
1412 LLM response generated by DeepSeek-V3 is as F-12

1413

1414

1415

1416

```
ASCC Check Result
{
  "discrepancies": [
    "The orthocenter is incorrectly defined as the vector sum of the triangle's vertices, violating the correct geometric definition of the orthocenter as the intersection point of the altitudes.", "The circumradius R is included in the MySystem structure but is not used in any constraints or the solution, despite being required in the original problem.", "The solution sol_sum_squares is calculated purely geometrically without being expressed in terms of a, b, c, and R as required by the original problem. Additionally, the orthocenter calculation is incorrect.", "The orthocenter H is not uniquely determined by the given constraints, violating the Exhaustion Principle of Predicate Satisfaction. The placeholder definition `sys.H = vec_add A (vec_add B C)` does not correctly compute the orthocenter."],
  "recommendations": [
    "Replace the placeholder definition of the orthocenter with a correct geometric definition. For example, use the formula for the orthocenter in terms of the vertices and the circumradius, or implement a function that computes the intersection point of the altitudes. The corrected definition should comply with the Mandatory Lean4 Template.", "Add a constraint that relates R to the triangle ABC, such as defining it as the circumradius of the triangle. For example, add a new constraint `circumradius_constraint` that computes R from the triangle's side lengths and angles or uses existing Mathlib functions to enforce this relationship.", "Modify the solution_constraint to express sol_sum_squares in terms of a, b, c, R using the known formula  $(9R^2 - (a^2 + b^2 + c^2)/2)$ . Also, implement a correct orthocenter calculation.", "Replace the placeholder definition with a correct computation of the orthocenter. For example, the orthocenter can be computed as the intersection point of the altitudes of the triangle. Here is a corrected definition:
```
lean4
def orthocenter_constraint (sys:MySystem) : Prop :=
  let ⟨A, B, C⟩ := sys.triangle
  let altitude_A := ... -- Compute the altitude from A to BC
  let altitude_B := ... -- Compute the altitude from B to AC
  sys.H = intersection_point altitude_A altitude_B -- Assuming a function to compute intersection points
```
Note: The exact implementation of the altitude and intersection point computations would depend on the available geometric functions in the Mathlib library."],
  "consistency_level": "level_3"
}
```

1447

1448

1449

Figure F-12: LLM response during ASCC check

1450

1451

1452

1453

1454

1455

1456

1457

1458  
1459  
1460  
1461  
1462  
1463  
1464**Problem:**

Let  $G$  and  $H$  denote the centroid and orthocenter of triangle  $ABC$ , respectively. Let  $F$  be the midpoint of  $\overline{GH}$ . Express  $AF^2 + BF^2 + CF^2$  in terms of the side lengths  $a, b, c$  and circumradius  $R$  of triangle  $ABC$ .

1465  
1466  
1467  
1468  
1469  
1470  
1471  
1472  
1473  
1474  
1475  
1476  
1477  
1478  
1479  
1480  
1481  
1482  
1483  
1484  
1485  
1486  
1487  
1488  
1489  
1490  
1491  
1492  
1493  
1494  
1495**Our method**

```

1  -->>declare_auxiliary_types
2  structure Triangle where
3    A : R^2
4    B : R^2
5    C : R^2
6  -->>declare_mathematical_system
7  structure MySystem where
8    triangle : Triangle
9    G : R^2
10   H : R^2
11   F : R^2
12   a : R
13   b : R
14   c : R
15   R : R
16   sol_sum_squares : R
17  -->>declare_auxiliary_functions_or_predicates
18  def vec_add (v : R^2) : R^2 := fun i => v i + w i
19  def vec_scale (s : R) (v : R^2) : R^2 := fun i => s * v i
20  def vec_div (v : R^2) (s : R) : R^2 := fun i => v i / s
21  def find_centroid (A B C : R^2) : R^2 := vec_div (vec_add A B) C 3
22  def find_midpoint (P Q : R^2) : R^2 := vec_div (vec_add P Q) 2
23  def dist_sq (P Q : R^2) : R :=
24    let dx := P 0 - Q 0
25    let dy := P 1 - Q 1
26    dx*dx + dy*dy
27  -->>declare_constraints
28  def triangle_constraint (sys:MySystem) : Prop :=
29  let (A, B, C) := sys.triangle
30  sys.a = Real.sqrt (dist_sq B C) ∧
31  sys.b = Real.sqrt (dist_sq A C) ∧
32  sys.c = Real.sqrt (dist_sq A B)
33  def centroid_constraint (sys:MySystem) : Prop :=
34  let (A, B, C) := sys.triangle
35  sys.G = find_centroid A B C
36  def orthocenter_constraint (sys:MySystem) : Prop :=
37  let (A, B, C) := sys.triangle
38  -- Simplified orthocenter calculation (placeholder)
39  sys.H = vec_add A (vec_add B C)
40  def midpoint_constraint (sys:MySystem) : Prop :=
41  sys.F = find_midpoint sys.G sys.H
42  def solution_constraint (sys:MySystem) : Prop :=
43  let (A, B, C) := sys.triangle
44  sys.sol_sum_squares = dist_sq A sys.F + dist_sq B sys.F + dist_sq C sys.F
45  -->>declare_the_problem_statement
46  def problem_statement (solution : R) : Prop :=
47  ∀ sys : MySystem,
48    triangle_constraint sys ∧
49    centroid_constraint sys ∧
50    orthocenter_constraint sys ∧
51    midpoint_constraint sys ∧
52    solution_constraint sys →
53    solution = sys.sol_sum_squares

```

Rectified to

```

1  -->>declare_auxiliary_types
2  structure Triangle where
3    A : R^2
4    B : R^2
5    C : R^2
6  -->>declare_mathematical_system
7  structure MySystem where
8    triangle : Triangle
9    G : R^2
10   H : R^2
11   F : R^2
12   a : R
13   b : R
14   c : R
15   R : R
16   sol_sum_squares : R
17  -->>declare_auxiliary_functions_or_predicates
18  def vec_sub (v w : R^2) : R^2 := fun i => v i - w i
19  def vec_scale (s : R) (v : R^2) : R^2 := fun i => v i * s
20  def vec_div (v : R^2) (s : R) : R^2 := fun i => v i / s
21  def find_centroid (A B C : R^2) : R^2 := vec_div (vec_add A B) C 3
22  def find_midpoint (P Q : R^2) : R^2 := vec_div (vec_add P Q) 2
23  def dist_sq (P Q : R^2) : R :=
24  let dx := P 0 - Q 0
25  let dy := P 1 - Q 1
26  dx*dx + dy*dy
27  def is_orthocenter (H : R^2) (A B C : R^2) : Prop :=
28  let AB := vec_sub B A
29  let AC := vec_sub C A
30  let BC := vec_sub C B
31  !(vec_sub H A), BC) = 0 ∧
32  !(vec_sub H B), AC) = 0 ∧
33  !(vec_sub H C), AB) = 0
34  def area_sq (a b c : R) : R :=
35  let s := (a + b + c)/2
36  let r := (a * b * c)/s
37  Real.sqrt (s*(s - a)*(s - b)*(s - c))
38  -->>declare_constraints
39  def triangle_constraint (sys:MySystem) : Prop :=
40  let (A, B, C) := sys.triangle
41  sys.a = Real.sqrt (dist_sq B C) ∧
42  sys.b = Real.sqrt (dist_sq A C) ∧
43  sys.c = Real.sqrt (dist_sq A B) ∧
44  sys.a > 0 ∧ sys.b > 0 ∧ sys.c > 0 ∧
45  sys.a + sys.b + sys.c > 0
46  def centroid_constraint (sys:MySystem) : Prop :=
47  sys.G = find_centroid sys.A sys.B sys.C
48  def orthocenter_constraint (sys:MySystem) : Prop :=
49  sys.H = find_orthocenter sys.A sys.B sys.C
50  def midpoint_constraint (sys:MySystem) : Prop :=
51  sys.F = find_midpoint sys.G sys.H
52  def circumradius_constraint (sys:MySystem) : Prop :=
53  sys.R := (sys.a * sys.b * sys.c) / (4 * area sys.a sys.b sys.c)
54  def solution_constraint (sys:MySystem) : Prop :=
55  let (A, B, C) := sys.triangle
56  let AB := dist_sq A B
57  let AC := dist_sq A C
58  let BC := dist_sq B C
59  let AF := dist_sq A sys.F
60  let BF := dist_sq B sys.F
61  let CF := dist_sq C sys.F
62  sys.sol_sum_squares = AF^2 + BF^2 + CF^2
63  -->>declare_the_problem_statement
64  def problem_statement (solution : R) : Prop :=
65  ∀ sys : MySystem,
66    triangle_constraint sys ∧
67    centroid_constraint sys ∧
68    orthocenter_constraint sys ∧
69    midpoint_constraint sys ∧
70    circumradius_constraint sys ∧
71    solution_constraint sys →
72    solution = sys.sol_sum_squares
73  end

```

1496  
1497  
1498  
1499  
1500  
1501  
1502  
1503  
1504  
1505  
1506  
1507  
1508  
1509  
1510  
1511**Baseline method**

```

1  theorem centroid_orthocenter_midpoint_problem (A B C G H F : R^2) (a b c R : R)
2    (solution : R)
3  (h_triangle : ∃ (A B C : R^2), A - B ∧ B - C ∧ C - A)
4  (h_centroid : G = (1/3 : R) • (A + B + C))
5  (h_orthocenter : let AB := B - A; let AC := C - A; let BC := C - B;
6    ((H - A), BC) = 0 ∧ ((H - B), AC) = 0 ∧ ((H - C), AB) = 0)
7  (h_midpoint : F = (1/2 : R) • (G + H))
8  (h_side_lengths : a = dist B C ∧ b = dist C A ∧ c = dist A B)
9  (h_circumradius : let AB := B - A; let AC := C - A;
10   R = dist A B * dist B C * dist C A / (2 * |(AB, AC)|))
11  : solution = (dist A F)^2 + (dist B F)^2 + (dist C F)^2 := by sorry

```

Figure F-13: Case study 3 to illustrate the rectification process compared to baseline

1512 APPENDIX G ADDITIONAL EXPERIMENTAL RESULTS  
1513

1514 To further eliminate potential biases from the dataset and the base model used for data generation,  
1515 we performed manual annotation on 50 samples randomly selected from the outputs of KIMI-K2 on  
1516 the miniF2F dataset. The sample set comprises 25 cases labeled as correct by ASCC-3-MV and 25  
1517 cases labeled as incorrect.

1518 The annotation results show that among the samples deemed correct by ASCC, all passed human  
1519 verification. In contrast, among those marked as incorrect by ASCC, 9 were judged as correct by hu-  
1520 man annotators. This finding aligns with the conclusions drawn from ASCC-Eval-150: the ASCC  
1521 method exhibits a low false positive rate and a relatively high but acceptable false negative rate.  
1522 These results, presented in Table 7, indicate that ASCC serves as a stringent evaluation criterion,  
1523 which is advantageous for our objective of identifying errors for feedback and correction. More-  
1524 over, they further affirm that the accuracy metric derived from ASCC-3-MV provides a reliable and  
1525 credible assessment.

1527 Table 7: Evaluation of ASCC-3-MV on results generated by KIMI-K2  
1528

Items	ASCC-positive	ASCC-negative
Human-positive	25	9
Human-negative	0	16

1533 We also conducted a comparative experiment using Bidirectional Equivalence(BEq) as the eval-  
1534 uation metric. The evaluation was limited to the LoC-Decomp method with KIMI-K2 as the  
1535 base model, along with the best-performing baseline in the LLM-as-a-judge setup, Goedel-  
1536 Autoformalizer-V2-32B. The results are summarized below in Table 8.

1538 Table 8: Evaluation using BEq on mini-F2F  
1539

Items	LoC-Decomp	Goedel-Autoformalizer-V2-32B
mini-F2F	62.82	54.91

1543 The BEq-based evaluation reveals notable differences in absolute pass rates compared to the LLM-  
1544 as-a-judge approach, which is expected given the probabilistic nature of LLM judgments and the  
1545 current limitations of automated theorem proving. Nevertheless, in terms of relative performance,  
1546 the BEq results align with those from LLM-as-a-judge: our method continues to show a clear ad-  
1547 vantage over fine-tuned specialized models.

1548 Detailed experimental setup: To perform BEq evaluation, it is necessary to formulate a bidirectional  
1549 implication theorem and supply its proof, as illustrated below:  
1550

1551 Code 2: Convert a theorem to it's Loc-Decomp counterpart and prove the equivalence  
1552

```
1553 theorem bidirectional_equivalence :  
1554   generated_proposition ↔ miniF2F_original_proposition := by
```

1555 Here, `miniF2F_original_proposition` denotes the original proposition from the miniF2F  
1556 dataset, while `generated_proposition` refers to the model-generated proposition. Since both  
1557 the miniF2F dataset and the Lean4 code produced by Goedel-autoformalizer-V2 are structured as  
1558 theorem statements awaiting proof, we first transformed them into predicate forms using the `def`  
1559 keyword. This conversion was carried out using the DeepSeek-V3.1 model.

1560 After constructing the `bidirectional_equivalence` theorem, we employed DeepSeek-V3.1  
1561 as an automated theorem prover. The prover was allowed up to 10 iterative corrections in case of  
1562 compilation errors. Only proofs that passed the compiler without errors were considered valid.  
1563

1564 We also report the self-check pass rates—where “self-check” refers to a single ASCC evaluation  
1565 performed by the current model itself during the iterative process, as opposed to DeepSeek-V3  
model usd in ASCC-3 metric—through one, two, and three rounds of iterative refinement on the

1566 MATH-500 and miniF2F datasets. The results show that the self-check pass rate increases with  
 1567 more iterations for both DeepSeek-V3 and KIMI-K2. In contrast, Qwen3-235B exhibits a significant  
 1568 deviation from our standard ASCC-3-MV metric. We argue this is because smaller-scale models  
 1569 struggle to make judgments on semantic consistency that align with human standards. See Figure  
 1570 G-14 for details.

1571

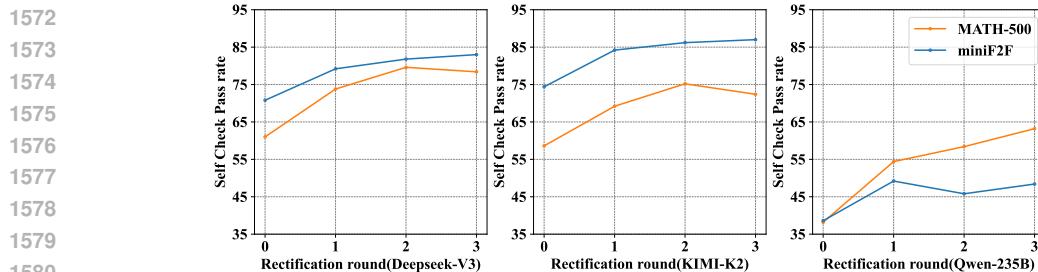


Figure G-14: Self-check pass rate during iteration

1581

1582

1583

1584

## 1585 APPENDIX H DETAILS ABOUT OUR BASELINES

1586

1587 For both the Formal-Baseline and Formal-Iterative-Baseline, we adopted exactly the same example  
 1588 set and few-shot mechanism as our approach, except that for the baseline method, the formal Lean  
 1589 4 code is transformed into a theorem with sorry like previous works(Azerbayev et al., 2023a; Li  
 1590 et al., 2024a; Wu et al., 2022). For the Formal-Iterative-Baseline, the iteration settings are exactly  
 1591 the same with our method, except it adopted a basic semantic checker. **The SFT-Baseline used the**  
 1592 **same configuration as the Formal-Baseline, with the exception that the base model was replaced by**  
 1593 **DeepSeek-Prover-V2.**

1594

1595

## APPENDIX I PROMPTS

1596

1597

## I.1 PROMPT DIFFERENCES ACROSS MODELS

1598

1599

1600

1601

1602

1603

1604

1605

1606

1607

1608

We observed that the Qwen3-235B model tends to generate overly lengthy responses (resembling a thinking mode despite being used as an instruct model). To address this, we introduced instructions such as *Your analysis must be concise but accurate.* and *Note: Avoid tediously long analysis.* to guide the model toward producing more concise outputs. The inclusion of these instructions successfully reduced verbosity in the model’s responses.

1609

1610

1611

1612

We further evaluated the effect of these instructions on DeepSeek-V3 and KIMI-K2. The results indicated that DeepSeek-V3 generated more concise content when the instructions were applied, which led to a performance drop on the ASCC evaluation using the MATH-150 dataset. In contrast, KIMI-K2 was largely unaffected by the instructions, demonstrating greater robustness to minor prompt variations.

1613

1614

1615

We argue that this discrepancy is reasonable, as differences in training strategies and data can lead to varied output styles across models. Although some minor differences on prompt is acceptable in practice, these findings highlight a limitation of our approach: it is susceptibility to specific prompt designs.

1616

1617

1618

1619

```
# Please use Lean 4 code to convert the textual description of the
  problem into a formal representation.
## Instruction
The following is a middle school mathematics problem. Please use Lean 4
  code to convert the textual description of the problem into a formal
  representation.
```

```

1620
1621 Any mathematical problem can be viewed as the study of the properties of
1622 a specific mathematical system. A mathematical system consists of
1623 several mathematical objects, along with a set of constraints that
1624 limit the properties or relationships of these objects, thereby
1625 defining the mathematical system under investigation. The
1626 requirement of the mathematical problem is to study a particular
1627 property of this mathematical system. Therefore, a mathematical
1628 problem can be formally expressed by declaring a mathematical system
1629 composed of several mathematical objects and a series of
1630 constraints, and for any concrete instance of the system, the
1631 conjunction of all these constraints imply a statement about the
1632 solution.
1633
1634 ## Attenions
1635 Note: You do not need to solve the problem, your task is solely to
1636 translate the problem's description into formal Lean 4 code.
1637 Note: For the naming of any symbols, if the stem explicitly provides a
1638 name, it should be consistent with the stem; otherwise, provide a
1639 reasonable name.
1640 Note: The sole purpose of naming is to improve code readability. Names
1641 cannot be relied upon to convey semantic meaning or mathematical
1642 structure all such information must be expressed through the code's
1643 logic.
1644 Note: Please pay attention to distinguishing the syntax differences
1645 between Lean 3 and Lean 4.
1646
1647 ## Problem
1648 The problem to be processed is as follows:
1649 {original_problem}
1650
1651 ## Response Template
1652 Here, you should formalize the above problem in Lean4, strictly
1653 following the provided Lean4 template as shown below.
1654
1655 ````Lean4
1656 import Mathlib
1657
1658 -->>declare_enviroment
1659 open Real InnerProductGeometry Matrix Topology Filter ENNReal Polynomial
1660 Classical Complex
1661
1662 notation " $\mathbb{R}^n \Rightarrow$  EuclideanSpace  $\mathbb{R}$  (Fin n)"
1663 notation " $M [m, n] \Rightarrow$  Matrix (Fin m) (Fin n)  $\mathbb{R}$ "
1664 notation " $\langle x, y \rangle \Rightarrow$  @inner  $\mathbb{R} \_ \_ x y$ "
1665
1666 variable {V : Type} [NormedAddCommGroup V] [InnerProductSpace  $\mathbb{R}$  V]
1667
1668 /-
1669 The foundational environment has been established, and you can not
1670 modify it. For the sake of automated control, declaring additional
1671 notations or opening more namespaces is not allowed.
1672 -/
1673
1674
1675 noncomputable section
1676 -->>declare_auxiliary_types
1677 -/
1678 Some mathematical objects have complex types and need to be declared
1679 using the 'structure' or 'inductive' keywords. At this stage, you
1680 can declare the types of certain complex mathematical objects for
1681 ease of later use.
1682 -/
1683
1684 -->>declare_mathematical_system
1685 -/

```

1674 In Lean 4, declaring a mathematical system involves defining a structure  
 1675 (must be named as 'MySystem' for automatic code check) composed of  
 1676 mathematical objects that may represent either structural spaces  
 1677 (possible value domains) or concrete instances (specific values).  
 1678 The mathematical system consists of several mathematical objects,  
 1679 along with a set of constraints that limit their properties or  
 1680 relationships. Follow the below principles when constructing this  
 1681 system:  
 1682

1. Complete System Declaration Principle:  
 - For readability, the mathematical system structure must contain all objects mentioned or implied in the problem statement, which enables immediate visual mapping between problem text and formal structure, and help the readers to understand.
2. Type Specification Principle:  
 - Declare only the types of mathematical objects at this stage.  
 - Detailed definitions and property relationships will be established later through constraint predicates.
3. Solution Object Principle:  
 - There must exists a special mathematical object that named with the prefix 'sol\_', denotes the target property/solution the problem seeks  
 - Analyze the problem to determine:  
 \* What constitutes a valid solution  
 \* Whether it represents a concrete value or solution space  
 \* The appropriate type representation (set/type for spaces, direct type for unique solutions)
4. Clarify Point-Space Principle  
 - A mathematical object should be formalized as a concrete instance (e.g.,  $a : \text{Nat}$ ) if and only if it is uniquely determined within the problem's constraints. Otherwise, it must be represented as a set or type (e.g.,  $\text{Set Nat}$ ) to preserve its possible value space.
5. Exhaustion Principle of Predicate Satisfaction  
 - If it is desired for set  $A$  to contain all elements that satisfy predicate  $P$ , it must be ensured that every element in set  $A$  satisfies predicate  $P$ , and that there does not exist an element  $e$  such that  $e$  satisfies predicate  $P$  but  $e$  does not belong to set  $A$ .  
 -/

1710 -->>declare\_auxiliary\_functions\_or\_predicates  
 1711 /-  
 1712 The definitions, properties, or relationships of declared objects need  
 1713 to be specified later by declaring predicate constraints. However,  
 1714 some predicates may be overly complex, so you can define auxiliary  
 1715 functions or predicates at this stage to simplify the subsequent  
 1716 predicate constraints. Predicates defined here cannot serve as  
 1717 antecedents in problem\_statement implications. They require  
 1718 encapsulation as constraint predicates in the next step before usage.  
 1719 For the purpose of automated control, parameters of type MySystem cannot  
 1720 be used in this step.  
 1721 -/

1722 -->>declare\_constraints  
 1723 /-  
 1724 In this step, you need to express the definitions, properties, or  
 1725 relationships of mathematical objects in the system by declaring  
 1726 predicates. The names of these predicates must end with  
 1727 '\_constraint' for better readability.

```

1728 All constraints from the original problem must be reflected in this
1729 step, and all constraint predicates declared here must appear in the
1730 problem statement as antecedents in implication expressions.
1731
1732 While maintaining strict logical rigor, prioritize code readability and
1733 semantic clarity by avoiding the conflation of multiple constraints
1734 within a single restrictive predicate. For the sake of code
1735 readability and semantic clarity, a reasonable amount of code
1736 redundancy is acceptable. Avoid sacrificing readability in pursuit
1737 of excessive brevity.
1738
1739 Additionally, the constraint describing the property of the 'sol_'
1740 object should be named 'solution_constraint', which is used to
1741 constrain the 'sol_' object so that it satisfies the requirements of
1742 the original problem and becomes the solution to it.
1743 -/
1744
1745 -->>declare_the_problem_statement
1746 -/
1747 In this step, you need to declare a special predicate called
1748 'problem_statement', which takes the 'solution' variable as a free
1749 input variable. This predicate applies all constraint predicates to
1750 the mathematical system, thereby completing the declarative
1751 expression of the original problem. The format of the
1752 'problem_statement' predicate is strictly defined: for any
1753 mathematical system, the following implication holds, where the
1754 antecedent is the logical AND of all constraint predicates, and the
1755 consequence is 'solution = sol_object'.
1756
1757 The semantics of 'problem_statement' are as follows: the proposition is
1758 true if and only if the value of 'solution' is the solution to the
1759 original problem.
1760 -/
1761
1762 end
1763
1764
1765 ## Examples

```

### I.3 THE PROMPT USED FOR BACK TRANSLATION (PER CONSTRIANTS)

```

1764
1765 ## Requirement : Translate Lean4 to natural language
1766 I am attempting to formalize a certain mathematical problem. To achieve
1767 this formalization, I model the mathematical problem as a
1768 mathematical system consisting of several mathematical objects and a
1769 series of constraints on the system. The following Lean4 code
1770 represents one of the complete formalized constraints for the
1771 mathematical problem. Additionally, MySystem describes the
1772 mathematical system, while the rest of the code serves as auxiliary.
1773 Please analyze the semantics of this constraint and express it in
1774 natural language. Pay attention the use of quantifiers and the domain
1775 of variables.
1776
1777 Any mathematical problem can be viewed as the study of the properties of
1778 a certain mathematical system. A mathematical system consists of
1779 several mathematical objects, along with a series of constraints
1780 that limit the properties or relationships of these mathematical
1781 objects, thereby defining the mathematical system under study. The
1782 requirement of a mathematical problem is to investigate a certain
1783 property of this mathematical system. Therefore, a mathematical
1784 problem can be formally expressed as declaring a mathematical system
1785 composed of several mathematical objects and a series of constraints.

```

```

1782
1783 Note: There is some distinctions between certain Lean4 data structures,
1784 for example, between List and Set, where a List may contain
1785 duplicate elements, while a Set enforces uniqueness, reflect the
1786 feature of the data structure used.
1787 Note: Your task is solely describe the semantic meaning of the
1788 constraint's Lean4 code, do not make any calculations or
1789 derivations. Focus exclusively on providing a precise mathematical
1790 characterization of the system formalized in this code. Do not make
1791 any assumptions about the real-world problem it might represent.
1792 Note: You must carefully translate the logical structure of Lean4 code,
1793 especially when it involves universal quantifiers or existential
1794 quantifiers.
1795 Note: The naming of objects is merely an identifier and should not be
1796 used as a reference for semantic meaning.
1797 Note: Describing Lean4 code in natural language may introduce ambiguity,
1798 so try your best to avoid ambiguity.
1799 Note: If any auxiliary functions are used in the Lean4 code of a
1800 constraint, describe their semantic meaning integrated naturally
1801 with the constraint, rather than referring to them by name.
1802
1803 Note: In your analysis and summary, you must reflect the below two
1804 principles (Clarify Point-Space Principle and Exhaustion Principle of
1805 Predicate Satisfaction).
1806 ## Formalization Principle
1807 Clarify Point-Space Principle
1808 - A mathematical object should be formalized as a concrete instance
1809 (e.g.,  $a : \text{Nat}$ ) if and only if it is uniquely determined within
1810 the problem's constraints. Otherwise, it must be represented as a
1811 set or type (e.g.,  $\text{Set Nat}$ ) to preserve its possible value space.
1812
1813 Exhaustion Principle of Predicate Satisfaction
1814 - If it is desired for set  $A$  to contain all elements that satisfy
1815 predicate  $P$ , it must be ensured that every element in set  $A$ 
1816 satisfies predicate  $P$ , and that there does not exist an element  $e$ 
1817 such that  $e$  satisfies predicate  $P$  but  $e$  does not belong to set  $A$ .
1818
1819 @magic_method_or_not
1820
1821 ## Your response should strictly follow the template below.
1822 ### **Analysis of '<constraint name>'**
1823 <Your analysis of the constraint, must reflect the above two principles>
1824
1825 ### **Concise Summary of '<constraint name>'**
1826 ````markdown
1827 <Here is a clear and natural language summary of the constraint's
1828 meaning that incorporates the semantics of any auxiliary functions.
1829 Emphasize adherence to the above principles, especially in cases
1830 where they may have been overlooked.>
1831
1832
1833
1834 ## Lean4 code you need to translate
1835 ````Lean4
1836 <Lean4_code>
1837
1838
1839
1840
1841 I.4 THE PROMPT USED FOR CONSISTENCY LEVEL DETERMINATION
1842
1843
1844 # Requirement: The Lean4 code is a formalization of
1845 problem_original(intendedly transformed into a verification task by
1846 introducing '{sol_obj}', since there is no concept such as 'solve' in

```

1836 Lean4), but it may have some issues in the formalization procedure.  
 1837 I have observed some semantic discrepancies between the Lean4 code  
 1838 and the problem\_original but I can not be sure whether these  
 1839 discrepancies are severe enough to break the semantic consistency.  
 1840 Please analyse every discrepancy listed and determine the  
 1841 consistency level of the Lean4 code, by the final criterion and  
 1842 definitions of consistency level. Your analyse must be concise but  
 1843 accurate.  
 1844 **## \*\*Consistency Criteria \*\***  
 1845 The Lean 4 code is deemed **\*\*consistent\*\*** with the original problem if  
 1846 and only if it **\*\*does not violate\*\*** any of the following  
 1847 **\*\*Violation Criteria\*\*** .  
 1848 **### \*\* Violation Criteria \*\***  
 1849 **\*(If any of these are true, the code is **\*\*inconsistent\*\***.)\***  
 1850 **\*\*Violation Criterion 1: Object Omission\*\***  
 1851 Not all mathematical objects in 'problem\_original' are reflected in the  
 1852 Lean4 code or their types mismatch(except for '{sol\_obj}', the type  
 1853 of '{sol\_obj}' can be different but must be equivalent). As long as  
 1854 the objects are correctly reflected, some redundancy in the Lean4  
 1855 code are acceptable.  
 1856 **\*\*Violation Criterion 2: Semantic Alteration\*\***  
 1857 The definitions, properties, or relationships of any mathematical object  
 1858 from 'problem\_original' are not exactly preserved in the Lean4 code.  
 1859 **\*(Note: Some redundancy is permitted as long as the core semantics  
 1860 are constrained correctly.)\***  
 1861 **\*\*Violation Criterion 3: Constraint Incompleteness\*\***  
 1862 The constraints expressed in problem\_formal fail to comprehensively  
 1863 represent all explicit and implicit constraints present in  
 1864 problem\_original. (Note: Full completeness is essential, both  
 1865 explicitly stated and implicitly inferred constraints must be  
 1866 accurately formalized.)  
 1867 **\*\*Violation Criterion 4: Over-Simplification\*\***  
 1868 The Lean4 code conducts concrete computation or derivation that  
 1869 simplifies the original problem, leading to a semantic inconsistency.  
 1870 **### \*\*Exceptions Criteria\*\***  
 1871 **\*\*Exception Criterion 1: Formatting Flexibility\*\***  
 1872 There is no need to strictly adhere to the original problem's formatting  
 1873 requirements for the solution, as Lean4 does not support extensive  
 1874 formatting options (such as requiring decimal numbers, etc.).  
 1875 **### \*\*Final Criterion \*\***  
 1876 Your determination should focus on the discrepancies listed, other  
 1877 aspects are irrelevant.  
 1878 The Lean 4 code is **\*\*consistent\*\*** with the original problem **\*\*if and  
 1879 only if\*\***: All the discrepancies listed are acceptable under the  
 1880 above criteria.  
 1881 As long as the core properties and relationships of the essential  
 1882 objects are preserved, some redundancy or unnecessary complexity in  
 1883 the Lean4 code is acceptable.  
 1884 **## Formalization Principle for reference:**  
 1885 Clarify Point-Space Principle  
 1886 - A mathematical object should be formalized as a concrete instance  
 1887 (e.g., a : Nat) if and only if it can be uniquely determined  
 1888 within the problem's constraints. Otherwise, it must be  
 1889 represented as a set or type (e.g., Set Nat) to preserve its  
 1890 possible value space.  
 1891 **Exhaustion Principle of Predicate Satisfaction**

```

1890      - If it is desired for set A to contain all elements that satisfy
1891          predicate P, it must be ensured that every element in set A
1892          satisfies predicate P, and that there does not exist an element e
1893          such that e satisfies predicate P but e does not belong to set A.
1894
1895      ## Mandatory Lean4 Template:
1896      ```Lean4
1897      -->>declare_enviroment
1898      <Some enviroment>
1899      -->>declare_auxiliary_types
1900      <some auxiliary types>
1901      -->>declare_mathematical_system
1902      -->>declare_mathematical_system
1903      structure MySystem where
1904          <objects>
1905          -->>declare_auxiliary_functions_or_predicates
1906          <some auxiliary definitions, without any parameter be of type MySystem>
1907          -->>declare_constraints
1908          def <name>_constraint (sys:MySystem) : Prop :=
1909              <expr_body>
1910          def <name>_constraint (sys:MySystem) : Prop :=
1911              <expr_body>
1912          def solution_constraint (sys:MySystem) : Prop :=
1913              <expr_body>
1914          -->>declare_the_problem_statement
1915          def problem_statement (solution : <type>) : Prop :=
1916              ∀ sys:MySystem,
1917                  <name>_constraint sys ∧
1918                  <name>_constraint sys ∧
1919                  solution_constraint sys →
1920                  <expression of solution and sol_object>
1921          end
1922      ```
1923
1924      ## problem_original:
1925      @original_problem
1926
1927      ## Lean4 Code To Judge:
1928      ```Lean4
1929      @Lean4_code
1930      ```
1931
1932
1933      ## Potential Sementic Discrepancies:
1934      @potential_sementic_discrepancies
1935
1936      ## Consistency level:
1937      There are three levels of consistency you can choose.
1938      level_1: Fully consistent by the final criterion.
1939      level_2: Consistent without loss of generality: The Lean4 code
1940          formalizes the problem by analyzing representative cases. In each of
1941          these cases, the final criterion are fully satisfied, and the
1942          general case follows through straightforward deduction. Or the
1943          formalization rely on equivalent conversions, such as transforming
1944          canonical equations into general form. In such cases, the
1945          formulation can be regarded as consistent without loss of generality.
1946      level_3: Inconsistent, any criterion breaked can lead to this.
1947
1948      Note: Your reasons and recommendations should avoid including any
1949          specific calculations or derivations, as this may lead to
1950          inconsistency.
1951
1952      Note: Avoid tediously long analysis.
1953      ## Your response should be like:
1954      ### Analyse of all the listed discrepancies:

```

```

1944 <Analyse the discrepancies by examining the Lean4 code and original
1945 problem to determine whether it is severe enough to impair the
1946 semantic consistency. Make sure every discrepancy listed is analysed
1947 in detail.>
1948
1949     ### Analyse Of each consistency level:
1950     <According to the definitions of consistency level and your above
1951     analyse about the discrepancies listed, determine which consistency
1952     level is appropriate. Your determination should focus on the
1953     discrepancies listed, other aspects are irrelevant.>
1954
1955     ### Consistency Level Determination
1956     ````json
1957     {
1958         "consistency_level" : "<level_1 or level_2 or level_3>",
1959         "discrepancies" : ["<According to your previous analysis, list all the
1960             semantic discrepancies here. If consistency_level is 'level_1',
1961             leave this field blank. Please provide some detailed descriptions of
1962             the discrepancies, ensuring it is concrete rather than high-level.
1963             Exclude discrepancies related to unnecessary abstractions,
1964             complexity, or redundancy, as that is not really matter.>"],
1965         "recommendations" : ["<Some specific recommendations to rectify the
1966             Lean4 code instead of high-level recommendations such as rectify
1967             which declaration into what. Your recommendation must align with the
1968             Formalization Basement and Mandatory Lean4 Template. The
1969             recommendations must directly correspond to the discrepancies. If
1970             consistency_level is 'level_1', leave this field blank.>"]
1971     }
1972     ````
```

## I.5 THE PROMPT USED FOR RECTIFICATION

```

1973 # Requirement: There appears to be a discrepancy between your Lean4
1974 formalization and the original problem, due to the following
1975 reasons, and some recommendations to rectify the Lean4 code are
1976 provided. Please rectify the Lean4 code accordingly. Print the
1977 rectified Lean4 code directly according to the error information. Do
1978 not make further thinking.
1979 Note: All suggestions are indicative, not mandatory. Rectify the Lean4
1980 code based on your understanding.
1981 Note: Your rectification must meet all the formalization principles
1982 mentioned above.
1983 Note: Your formalization must adopt this treatment: the solution to the
1984 original problem is taken as a free variable in the proposition
1985 **problem_statement**, such that **problem_statement** holds true if
1986 and only if the **solution** is indeed a valid solution to the
1987 original problem.
1988
1989 The determination of consistency follows the criteria below.
1990 ## **Consistency Criteria**
1991 The Lean 4 code is deemed **consistent** with the original problem if
1992 and only if it **does not violate** any of the following
1993 **Violation Criteria**.
1994 ### ** Violation Criteria ***
1995 *(If any of these are true, the code is **inconsistent**.)*
1996 **Violation Criterion 1: Object Omission**
1997 Not all mathematical objects in 'problem_original' are reflected in the
1998 Lean4 code or their types mismatch(except for '{sol_obj}', the type
1999 of '{sol_obj}' can be different but must be equivalent). As long as
2000 the objects are correctly reflected, some redundancy in the Lean4
2001 code are acceptable.
2002
2003 **Violation Criterion 2: Semantic Alteration**
```

1998 The definitions, properties, or relationships of any mathematical object  
 1999 from 'problem\_original' are not exactly preserved in the Lean4 code.  
 2000 \* (Note: Some redundancy is permitted as long as the core semantics  
 2001 are constrained correctly.)\*  
 2002  
 2003 **\*\*Violation Criterion 3: Constraint Incompleteness\*\***  
 2004 The constraints expressed in problem\_formal fail to comprehensively  
 2005 represent all explicit and implicit constraints present in  
 2006 problem\_original. (Note: Full completeness is essential both  
 2007 explicitly stated and implicitly inferred constraints must be  
 2008 accurately formalized.)  
 2009  
 2010 **\*\*Violation Criterion 4: Over-Simplification\*\***  
 2011 The Lean4 code conducts concrete computation or derivation that  
 2012 simplifies the original problem, leading to a semantic inconsistency.  
 2013  
 2014 **### \*\*Exceptions Criteria\*\***  
 2015 **\*\*Exception Criterion 1: Formatting Flexibility\*\***  
 2016 There is no need to strictly adhere to the original problem's formatting  
 2017 requirements for the solution, as Lean4 does not support extensive  
 2018 formatting options (such as requiring decimal numbers, etc.).  
 2019  
 2020 **### \*\*Final Criterion \*\***  
 2021 Your determination should focus on the discrepancies listed, other  
 2022 aspects are irrelevant.  
 2023 The Lean 4 code is **\*\*consistent\*\*** with the original problem **\*\*if and**  
 2024 **only if\*\***: All the discrepancies listed are acceptable under the  
 2025 above criteria.  
 2026 As long as the core properties and relationships of the essential  
 2027 objects are preserved, some redundancy or unnecessary complexity in  
 2028 the Lean4 code is acceptable.  
 2029  
 2030 Reflect the Clarify Point-Space Principle and the Exhaustion Principle  
 2031 of Predicate Satisfaction in your analysis. For some cases,  
 2032 violation of these principles may lead to severe semantic  
 2033 inconsistencies.  
 2034  
 2035 Clarify Point-Space Principle  
 2036 - A mathematical object should be formalized as a concrete instance  
 2037 (e.g.,  $a : \text{Nat}$ ) if and only if can be uniquely determined within  
 2038 the problem's constraints. Otherwise, it must be represented as a  
 2039 set or type (e.g.,  $\text{Set Nat}$ ) to preserve its possible value space.  
 2040  
 2041 Exhaustion Principle of Predicate Satisfaction  
 2042 - If it is desired for set  $A$  to contain all elements that satisfy  
 2043 predicate  $P$ , it must be ensured that every element in set  $A$   
 2044 satisfies predicate  $P$ , and that there does not exist an element  $e$   
 2045 such that  $e$  satisfies predicate  $P$  but  $e$  does not belong to set  $A$ .  
 2046  
 2047 Final criterion:  
 2048 The Lean 4 code is consistent with the original problem if and only if,  
 2049 for any concrete instance of MySystem satisfying the constraints,  
 2050 the three basic criteria are satisfied. As long as the properties or  
 2051 relationships of those essential objects are not compromised, some  
 2052 redundancy or unnecessary complexity in the Lean4 code is acceptable  
 2053 as long as the above three criteria are fully satisfied.  
 2054  
 2055 **## Reasons:**  
 2056 @reasons  
 2057  
 2058 **## Recomendations:**  
 2059 @recommendations  
 2060  
 2061 @previous\_recommendations

```

2052 ## Your response should strictly follow the following template:
2053
2054 ```Lean4
2055 import Mathlib
2056
2057 -->>declare_enviroment
2058 open Real InnerProductGeometry Matrix Topology Filter ENNReal Polynomial
2059   Classical Complex
2060
2061 notation " $\mathbb{R}^n \Rightarrow$  EuclideanSpace  $\mathbb{R}$  (Fin n)
2062 notation " $M[m, n] \Rightarrow$  Matrix (Fin m) (Fin n)  $\mathbb{R}$ 
2063 notation " $\langle x, y \rangle \Rightarrow$  @inner  $\mathbb{R} \_ \_ x y$ 
2064
2065 variable {V : Type} [NormedAddCommGroup V] [InnerProductSpace  $\mathbb{R}$  V]
2066
2067 /-
2068 The foundational environment has been established, and you can not
2069   modify it. For the sake of automated control, declaring additional
2070   notations or opening more namespaces is not allowed.
2071 The purpose of this formalization project is for educational purposes,
2072   so we strive to minimize reliance on implementations from the
2073   Mathlib library. Apart from 'import Mathlib', no additional imports
2074   should be included.
2075 -/
2076
2077 noncomputable section
2078 -->>declare_auxiliary_types
2079 <some auxiliary types>
2080 -->>declare_mathematical_system
2081 structure MySystem where
2082   <objects>
2083 -->>declare_auxiliary_functions_or_predicates
2084 <some auxiliary definitions, without any parameter be of type MySystem>
2085 -->>declare_constraints
2086 def <name>_constraint (sys:MySystem) : Prop :=
2087   <expr_body>
2088 def <name>_constraint (sys:MySystem) : Prop :=
2089   <expr_body>
2090 def solution_constraint (sys:MySystem) : Prop :=
2091   <expr_body>
2092 -->>declare_the_problem_statement
2093 def problem_statement (solution : <type>) : Prop :=
2094    $\forall$  sys:MySystem,
2095     <name>_constraint sys  $\wedge$ 
2096     <name>_constraint sys  $\wedge$ 
2097     solution_constraint sys  $\rightarrow$ 
2098     <expression of solution and sol_object>
2099   end
2099 ```

```

## I.6 THE PROMPT USED FOR SC-BASELINE

```

2098 You will receive a natural language math problem statement, along with
2099   its formal statement
2100   in LEAN 4 and, in some cases, a description of mathematical terms.
2101   Please evaluate whether
2102   the formal LEAN statement appropriately translates the natural language
2103   statement based on
2104   the following criteria. They are considered different if any of the
2105   criteria are not satisfied.
2106 1. Key Elements: The fundamental mathematical components, including
2107   variables,

```

```

2106 constants, operations, domain, and codomain are correctly represented in
2107 LEAN code.
2108 2. Mathematical Accuracy: The mathematical relationships and expressions
2109 should be
2110 interpreted consistently during translation.
2111 3. Structural Fidelity: The translation aligns closely with the original
2112 problem, maintaining
2113 its structure and purpose.
2114 4. Comprehensiveness: All conditions, constraints, and objectives stated
2115 in the natural
2116 language statement are mathematically included in the LEAN translation.
2117 When doing evaluation, break down each problem statement into
2118 components, match the
2119 components, and evaluate their equivalence.
2120 Think step-by-step and explain all of your reasonings.
2121
2122 Natural language math problem statement:
2123 {original_problem}
2124
2125 Formal statement in LEAN 4
2126 {Lean4_code}
2127 Your answer should be like:
2128 <some analyase here>
2129
2130 Judgement and recommendation:
2131 ```json
2132 {{{
2133     "consistent_or_not" : "<yes or no>",
2134     "reasons" : "<if inconsistent, leave your reasons here>",
2135     "recommendations" : "<if inconsistent, leave your recommendations
2136         here>"}
2137 }}}
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4280
4281
4282
4283
4284
4285
4286
4287
4288
4289
4290
4291
4292
4293
4294
4295
4296
4297
4298
4299
4300
4301
4302
4303
4304
4305
4306
4307
4308
4309
4310
4311
4312
4313
4314
4315
4316
4317
4318
4319
4320
4321
4322
4323
4324
4325
4326
4327
4328

```

```

2160 Judgement: [Your Answer, one of Appropriate, Inappropriate]
2161
2162 Natural language math problem statement:
2163 {original_problem}
2164
2165 Formal statement in LEAN 4
2166 {Lean4_code}
2167
2168 Natural language description of LEAN 4:
2169 {back_translation}
2170

```

## I.8 THE PROMPT USED FOR FORMAL-BASELINE

```

2173 Please translate the following mathematical problem into Lean4 by
2174 completing the template provided. Do not try to solve the problem,
2175 your task is formalization. Do not make any derivation or comutation
2176 as that would be inconsistent with the original problem. Enclose
2177 your Lean4 code in a ```Lean4``` section.

```

```

2178 Problem:
2179 {problem}
2180
2181 Lean4 tamplate to complete:
2182 ```Lean4
2183 import Mathlib
2184
2185 open Real InnerProductGeometry Matrix Topology Filter ENNReal Polynomial
2186 Classical Complex
2187
2188 notation " $\mathbb{R}^n \Rightarrow$  EuclideanSpace  $\mathbb{R}$  (Fin n)"
2189 notation " $M[m, n] \Rightarrow$  Matrix (Fin m) (Fin n)  $\mathbb{R}$ "
2190 notation " $\langle x, y \rangle \Rightarrow$  @inner  $\mathbb{R}^{m \times n} x y$ "
2191
2192 variable {V : Type} [NormedAddCommGroup V] [InnerProductSpace  $\mathbb{R}$  V]
2193
2194 noncomputable section
2195 theorem <name> <parameters> (solution:<type of solution>) : <conclusion>
2196   := by sorry
2197
2198 end
2199
2200 Examples:
2201
2202 ```

```

## I.9 THE PROMPT USED FOR FORMAL-ITERATIVE-BASELINE(semantic rectification)

```

2204
2205 # Your Lean4 code is not consistent with the original problem for the
2206 # following reasons, and try to rectify your Lean4 code according to
2207 # the reasons and recommendations.
2208
2209 ## Reasons:
2210 {reasons}
2211
2212 ## Recommendations:
2213 {recommendations}
2214
2215 Lean4 tamplate to complete:

```

```

2214 ````Lean4
2215 import Mathlib
2216
2217 open Real InnerProductGeometry Matrix Topology Filter ENNReal Polynomial
2218   Classical Complex
2219
2220 notation " $\mathbb{R}^n$ " => EuclideanSpace ℝ (Fin n)
2221 notation " $M[m, n]$ " => Matrix (Fin m) (Fin n) ℝ
2222 notation " $\langle x, y \rangle$ " => @inner ℝ _ _ x y
2223
2224 noncomputable section
2225 theorem <name> <parameters> (solution:<type of solution>) : <conclusion>
2226   := by sorry
2227
2228 end
2229
2230
2231 I.10 THE PROMPT USED FOR FORMAL-ITERATIVE-BASELINE(SYNTACTIC RECTIFICATION)
2232
2233
2234 # Requirement: There appears to be some errors in your Lean4 code.
2235   Please rectify the Lean4 code accordingly.
2236 Note: You only need to correct syntax errors, do not change any
2237   declaration's name, parameters, type, or semantic, otherwise the
2238   Lean4 code will be inconsistent with the original problem.
2239 Note: Try to implement the same logic using simpler syntax by yourself,
2240   rather than relying on Lean4 or Mathlib's built-in special features,
2241   to reduce the possibility of syntax errors or compilation errors.
2242
2243 ## Errors:
2244 @err_msg_str
2245
2246
2247
2248 ## Your response must follow the following template:
2249
2250 ````Lean4
2251 import Mathlib
2252
2253 open Real InnerProductGeometry Matrix Topology Filter ENNReal Polynomial
2254   Classical Complex
2255
2256 notation " $\mathbb{R}^n$ " => EuclideanSpace ℝ (Fin n)
2257 notation " $M[m, n]$ " => Matrix (Fin m) (Fin n) ℝ
2258 notation " $\langle x, y \rangle$ " => @inner ℝ _ _ x y
2259
2260 noncomputable section
2261 theorem <name> <parameters> (solution:<type of solution>) : <conclusion>
2262   := by sorry
2263
2264 end
2265
2266
2267

```