

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 DIVIDE AND ABSTRACT: AUTOFORMALIZATION VIA DECOMPOSITION AND ABSTRACTION LEARNING

Anonymous authors

Paper under double-blind review

ABSTRACT

Autoformalization, the task of translating informal mathematics into formal machine-verifiable languages, remains challenging due to three fundamental limitations: bottleneck by existing abstractions, difficulty of complex statements, and poor transferability across languages. Existing approaches rely heavily on pre-defined libraries and expect LLMs to directly generate complete formalizations, limiting their generalizability to complex statements and new formal languages. We propose *Divide and Abstract (DNA)*, a zero-training framework that addresses these challenges through a two-phase approach. First, *DNA* extracts common mathematical concepts from the entire corpus and formalizes them as reusable abstractions, extending the target language’s capability. Second, *DNA* hierarchically decomposes each statement into structured informal clauses, translates each clause using the learned abstractions, and composes them into complete formalizations. Our evaluation on the LeanEuclidPlus and ProofNet-Hard benchmarks demonstrates consistent improvements across multiple model families, achieving up to $8.60\times$ performance gains over baseline approaches. Notably, *DNA* enables smaller models to match larger baseline models and shows particularly strong performance on complex mathematical statements requiring nested reasoning. The framework requires no training on target languages, making it effective for low-resource domain-specific languages. Our code is available at <https://github.com/anonymousauthor567/DivdedAndAbstract>.

1 INTRODUCTION

Autoformalization is the task of automatically translating informal mathematics into formal languages designed for theorem provers such as Lean (Moura & Ullrich, 2021) and Z3 (De Moura & Bjørner, 2008), so that theorems and proofs written in natural languages can be mechanically verified. The success of autoformalization will significantly reduce the amount of labor from human experts, and thus empower three transformative applications:

1. Synthesizing informal-formal parallel data for training neural theorem provers (Xin et al., 2024).
2. Grounding and guiding reasoning processes in natural languages (Yang et al., 2022).
3. Accelerating verification of important mathematical theorems (Gonthier et al., 2013; Hales et al., 2015) or engineering systems (Zhao et al., 2012; Reid et al., 2016) not only to identify potential gaps and mistakes, but also to facilitate certified extensions and future theoretical discovery.

Inspired by the breakthrough of Large Language Models (LLMs) in machine translation between natural languages, researchers have increasingly applied LLMs to autoformalization (Weng et al., 2025). Even though the ultimate vision is to automatically formalize entire theories that include axioms, definitions, notations, theorems, and proofs, current LLMs struggle to even formalize an individual statement (Murphy et al., 2024; Liu et al., 2025a).

Previous work in LLM-based statement autoformalization has explored in-context learning (Wu et al., 2022), fine-tuning specialized autoformalizer models (Jiang et al., 2023a), invoking retrieval before LLM inference (Liu et al., 2025a), or employing post-inference techniques such as majority voting and semantic consistency ranking by an embedding model (Li et al., 2024a). However, all these approaches suffer from limited generalizability due to three fundamental challenges:

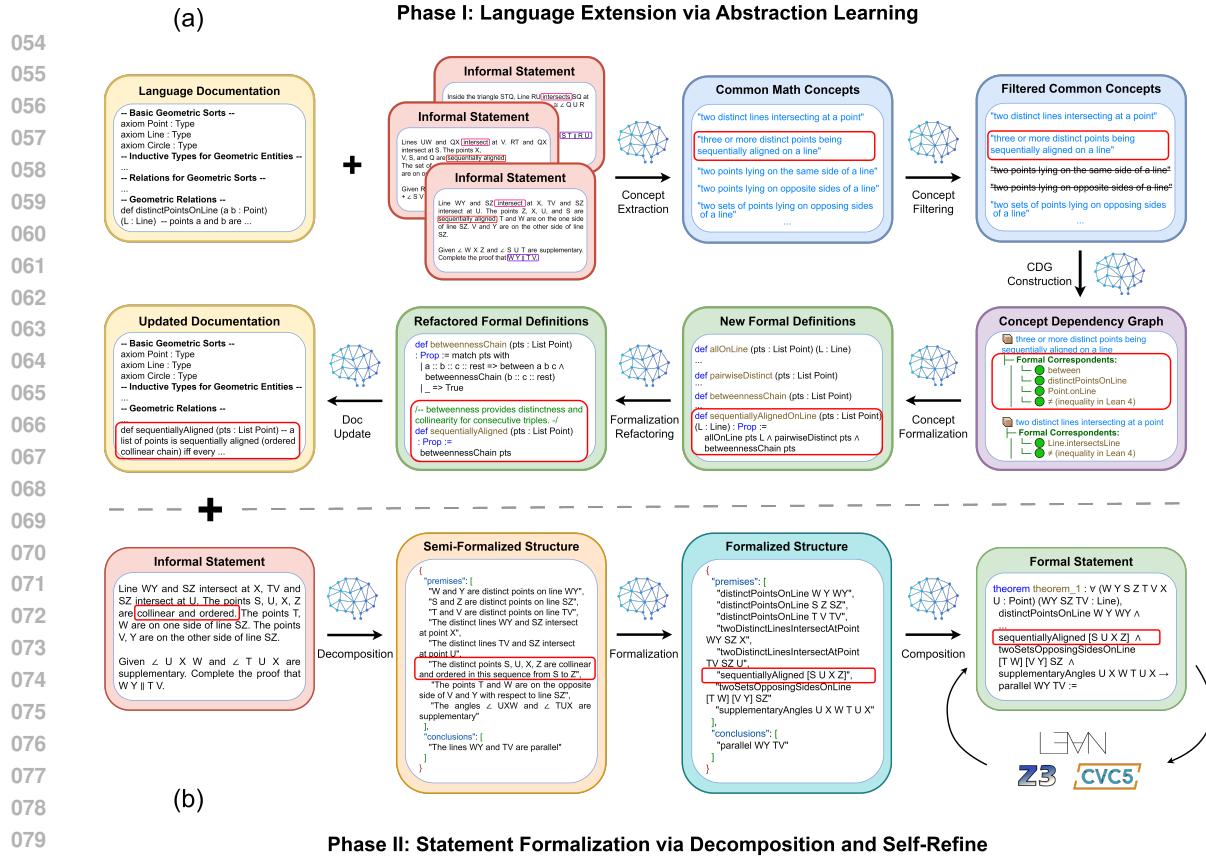


Figure 1: Overview of the *Divide and Abstract (DNA)* framework. **(a) Phase I (Language Extension):** Given an informal corpus, we extract common mathematical concepts, filter out those already present in the target language, construct a concept dependency graph (CDG), and systematically formalize concepts following topological ordering to extend the DSL. **(b) Phase II (Statement Formalization):** For each statement, we hierarchically decompose it into semi-formalized components, translate each component using the learned abstractions, compose them into complete formal statements, and refine using symbolic validator feedback.

1. Bottleneck by Existing Abstractions: The performance of formalizing the same informal statement is limited by the richness of existing abstractions in the target formal language. For example, if the target language only contains basic formal definitions of sets and real numbers, then formalizing the relation “a topological space being a manifold” would be difficult even for human experts. In contrast, if the target language already provides a formal relation “isManifold”, then the translation is simply a one-to-one mapping. However, all existing methods rely heavily on predefined abstractions in existing libraries like Mathlib in Lean (Mathlib Community, 2020) and HOL sessions in Isabelle (Nipkow et al., 2002), which fail to generalize to the cases where the target language doesn’t provide an extensive coverage of high-level abstractions.

2. Difficulty of Complex Statements: Regardless of the availability of relevant abstractions, the difficulty of formalization also grows proportionally to the complexity of statements. For example, formalizing a simple statement like “function f is continuous” is straightforward when the formal definition of continuity exists. However, formalizing a nested statement such as “the space of continuous functions from a compact manifold to a Banach space forms a complete metric space under the supremum norm” requires understanding and correctly composing multiple layers of mathematical abstractions. Nonetheless, all existing approaches expect the LLMs to directly generate the final formalization at inference time, which does not generalize well to complex statements that involve nested quantifiers, higher-order objects, and composite relations.

3. Transferability to Different Languages: Orthogonal to abstraction availability and statement complexity, the choice of target formal language constitutes a third dimension of the generalizability challenge. Most fine-tuned autoformalizer models (Gao et al., 2025; Wang et al., 2025; Xuejun

108 et al., 2025) are heavily trained on Lean Mathlib and formal statements that directly use Mathlib
 109 abstractions. However, as shown in Table 5, these models perform poorly when the target is a
 110 domain-specific language (DSL) that is not in the training data, or even when Mathlib evolves to
 111 newer versions with updated syntax and library structures.

112 To address these limitations in generalizability, we propose an end-to-end framework *Divide and*
 113 *Abstract (DNA)* with three key designs that directly correspond to each challenge:
 114

115 **1. Abstraction Learning:** While previous methods treat the formalization of each statement in-
 116 dependently, we leverage the insight that statements within the same corpus often share common
 117 mathematical concepts like definitions of objects and relations. As shown in Figure 1(a), given the
 118 current language documentation and a target informal corpus, *DNA* first extracts these shared con-
 119 cepts from the entire corpus and formalizes them as a collection of reusable formal abstractions.
 120 This phase essentially extends the target language by enriching the library of available abstractions,
 121 thereby facilitating the statement formalization both within and beyond the corpus.

122 **2. Hierarchical Decomposition:** Unlike existing approaches that let LLMs directly generate the
 123 formal statement, we take advantage of the fact that mathematical statements are semi-structured
 124 data — a statement consists of the quantifications and the body, the body can be broken down
 125 into a list of premises and a list of conclusions, and quantified statements nested in the premises
 126 or conclusion can be recursively decomposed in the same manner. As shown in Figure 1, *DNA*
 127 hierarchically decomposes a statement into a structure of informal clauses, translates each clause
 128 into its formal correspondents, and composes the formal clauses back into a complete formalization.
 129 Since we have learned a rich library of reusable abstractions, the formalization of each individual
 130 clause becomes significantly more tractable than formalizing the entire complex statement at once.

131 **3. Zero-Training Framework:** *DNA* is designed to be a zero-training, plug-and-play framework.
 132 This is achieved by simply providing the target language’s documentation in the context of the LLM.
 133 After the abstraction learning phase, the documentation is updated with the newly learned formal
 134 definitions, and then passed as context to the statement formalization phase, conditioning the LLM
 135 on both the original language specifications and a library of directly usable high-level definitions.
 136 The entire framework requires no training on the target formal language, and is thus particularly
 137 effective for low-resource DSLs.

138 We evaluate *DNA* on LeanEuclidPlus and ProofNet-Hard against fine-tuned autoformalizers and
 139 general-purpose LLMs including GPT-4.1/5, Claude-4-Sonnet, and Qwen3 variants. *DNA* consis-
 140 tently outperforms baselines across all model families, with the most dramatic improvement being
 141 Qwen3-14B advancing from 1.0 to 9.6 success rate (9.6 \times gain), while GPT-4.1-mini improves from
 142 4.8 to near GPT-4.1 levels. On ProofNet-Hard, all baselines achieve zero success while *DNA* enables
 143 successful formalization, demonstrating its effectiveness for domain-specific languages lacking ex-
 144 tensive training data. To summarize, our offer the following contributions in this paper:
 145

146 **(1)** identifying three generalizability challenges in autoformalization, **(2)** proposing a zero-training
 147 framework combining abstraction learning and hierarchical decomposition, **(3)** demonstrating 1.17 \times
 148 to 9.6 \times improvements across diverse models and benchmarks, and **(4)** enabling effective transfer to
 149 challenging domain-specific languages where specialized models fail entirely.

2 THE *Divide and Abstract* FRAMEWORK

2.1 PHASE I: LANGUAGE EXTENSION

150 As illustrated in Figure 1(a), the language extension process unfolds through six coordinated steps.
 151 The first three steps focus on abstraction learning and dependency analysis, while the latter three
 152 handle engineering concerns to transform the learned concepts into a downstream-user-friendly li-
 153 brary of plug-and-play abstractions.

154 **Step 1: Concept Extraction.** Given the informal corpus as input, we systematically extract common
 155 mathematical concepts using an LLM. We identify three types of concepts: definitions of mathemat-
 156 ical objects such as points, groups, and series; relations between objects such as “a point being on a
 157 line” or “a group being a subgroup of another”; and functions mapping objects to objects such as Eu-
 158 clidean distance or determinant. Our extraction process enforces precision by specifying argument

162
163
164
165
166
167
168
169
170
171

Step 1: Concept Extraction			Step 2: Concept Filtering	
Run	Recall	Correctness	Run	Correctness
Average	94.44%	100.00%	Average	100.00%

Table 1: Performance analysis for concept extraction and filtering steps.

types and counts, well-definedness by ensuring conventional mathematical validity, and abstractness by avoiding particular variable names or specific instances.

Step 2: Concept Filtering. Using the extracted concepts and current DSL documentation as input, we filter out concepts that already have direct formal correspondents in the target language. This prevents reinventing existing abstractions while focusing on genuinely missing concepts that would extend the language’s expressiveness. The filtering process leverages LLM understanding of both the extracted concepts and existing DSL capabilities to identify gaps in abstraction coverage.

Algorithm 1 Concept Dependency Graph (CDG) Construction Algorithm.

Input: Filtered Concepts C , Current Language Documentation D , LLM M

Output: Concept Dependency Graph G containing all concepts in C and their dependencies

```

1: Initialize the analysis result dict  $\mathcal{A} \leftarrow \emptyset$ 
2: Initialize the to-be-analyzed queue  $\mathcal{Q} \leftarrow C$ 
3: Initialize the have-analyzed set  $\mathcal{S} \leftarrow \emptyset$ 
4: while  $\mathcal{Q} \neq \emptyset$  do
5:    $c \leftarrow \mathcal{Q}.\text{dequeue}()$                                  $\triangleright$  fetch the next to-be-analyzed concept
6:   Analyze concept  $c$  using LLM  $M$  given context  $(D, \mathcal{A})$ , determine the dependency status of
7:    $c$ , which can be one of: “directly_expressible”, “needs_dependencies”, or “impossible”
8:    $\mathcal{A}[c] \leftarrow$  analysis result                                 $\triangleright$  save analysis result
9:    $\mathcal{S} \leftarrow \mathcal{S} \cup \{c\}$                                           $\triangleright$  add  $c$  to the have-analyzed set
10:  if  $c$  has status “needs_dependencies” then
11:    for each dependency  $d$  in the dependency list of  $c$  do
12:      if  $d \notin \mathcal{S}$  and  $d \notin \mathcal{Q}$  then
13:         $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{d\}$                                           $\triangleright$  add  $d$  to the to-be-analyzed queue
14:  return  $\mathcal{A}$ 

```

Step 3: Concept Dependency Graph (CDG) Construction. This crucial step builds a dependency graph mapping relationships between filtered concepts. For each concept, we analyze direct expressibility using existing DSL elements, prerequisite concept dependencies, and formalization feasibility. The CDG contains two node types: leaf nodes that are either directly formalizable or impossible to formalize within the target language scope, and parent nodes that depend on other concepts. Formalization follows topological ordering by starting with leaf nodes, then proceeding layer by layer up the dependency hierarchy. Algorithm 1 details the construction process, which includes safeguards against circular dependencies and convergence criteria. In our experiments on LeanEuclidPlus and ProofNet-Hard, most extracted concepts are directly formalizable, with only occasional multi-layer dependencies. Notably, CDG construction inherently performs concept decomposition, demonstrating that abstraction and decomposition are complementary—neither achieves optimal performance in isolation (Section 4).

207
208
209
210
211

Step 3: CDG Construction		Step 4: Concept Formalization	
Run	Correctness	Run	Correctness
Average	100.00%	Average	97.87%

Table 2: Performance analysis for dependency graph construction and concept formalization steps.

212
213
214
215

Step 4: Concept Formalization. With the CDG providing a clear formalization roadmap, we systematically formalize concepts following topological ordering. Starting with leaf nodes that are directly expressible using existing DSL elements, we proceed layer by layer through the dependency

216 hierarchy. The formalization process adheres to target language conventions and ensures syntactic
 217 correctness through iterative refinement.

218 **Step 5: Formalization Refactoring.** Due to the large number of concepts requiring formalization,
 219 we process them in batches, which may introduce repetitive content or overly specialized definitions.
 220 This step refactors the formalized concepts to eliminate redundancy and remove unnecessarily spe-
 221 cialized abstractions that would confuse downstream users. The goal is to produce a clean, coherent
 222 library of abstractions with clear, distinct APIs.

Step 5: Refactoring			Step 6: Documentation		
Run	Correctness	Compression	Correctness	Qwen3-235B	GPT-5 mini
Average	99.15%	1.59	100.00%	97.4%	99.8%

223 Table 3: Performance analysis for refactoring and documentation update steps in Phase I.

224 **Step 6: Documentation Update.** To ensure usability for downstream applications, we generate
 225 comprehensive documentation that excludes implementation details and private helper functions.
 226 The documentation focuses on high-level mathematical semantics, providing clear explanations of
 227 what each formal definition represents mathematically without exposing internal complexity.

228 For Phase I language extension, we employ Qwen3-235B-Instruct for concept extraction, filtering,
 229 and dependency graph construction on LeanEuclidPlus, while using GPT-5 with high reasoning
 230 effort for ProofNet-Hard. For the more complex tasks of concept formalization, refactoring, and
 231 documentation generation, we utilize GPT-5 with high reasoning effort across both benchmarks to
 232 ensure high-quality formal definitions and coherent abstraction libraries.

233 2.2 PHASE II: STATEMENT FORMALIZATION

234 Building upon the extended DSL from Phase I, Phase II employs hierarchical decomposition to
 235 formalize individual statements. Unlike existing approaches that expect LLMs to directly generate
 236 complete formalizations, our framework systematically breaks down complex statements into
 237 manageable components that can be formalized using the learned abstractions.

238 **Motivation and Error Analysis.** To motivate our decomposition approach, we conducted an error
 239 analysis on 200 failed formalization attempts (100 each from Qwen3-235B and GPT-5) using base-
 240 line methods without our pipeline. As shown in Table 4, we identified four primary error categories.
 241 Validation errors include syntax issues and logical consistency violations that prevent successful
 242 compilation. Stronger or weaker translation errors represent semantic misalignment where formal-
 243 izations are either too restrictive or too permissive compared to the English statement. Incorrect
 244 translation errors involve wrong formal correspondents or parameters for relations. Unfaithful vari-
 245 able naming creates inconsistency between variable names in formal and natural language. This
 246 analysis reveals distinct error patterns across models, motivating our decomposition approach that
 247 systematically addresses each error type.

Error Type	Qwen3-235B	GPT-5
Unfaithful Variable Name	1	1
Validation Error	8	2
Stronger/Weaker Translation	10	17
Incorrect Translation	11	8

262 Table 4: Error analysis of 200 failed formalization attempts, where 100 are from Qwen3-235B and
 263 another 100 from GPT-5. The numbers shown are the count for each error type. Four main error
 264 categories were identified.

265 **Solution for Each Error Type.** Our four-step decomposition pipeline systematically addresses each
 266 significant error type, excluding “Unfaithful Variable Name”, through targeted design choices. Step
 267 1 decomposes complex informal statements into semi-formalized structures with explicit quantifi-
 268 cations, premises, and conclusions, ensuring logical scope precisely matches the informal statement
 269 and addressing stronger/weaker translation errors. Step 2 translates each component using learned

270 abstractions from Phase I, enabling focused attention on correct formal correspondents and parameter assignments for individual clauses rather than managing entire statement complexity, thereby addressing incorrect translation errors. For instance, in LeanEuclidPlus, the relation `formTriangle` requires specific parameter ordering where points a, b must lie on line l_1 , points b, c on line l_2 , and points a, c on line l_3 —our clause-by-clause approach helps prevent logical contradictions from incorrect parameter ordering.

271 Step 3 systematically composes formalized components back into complete formal statements while
 272 ensuring proper quantifier scoping and logical consistency, reducing validation errors through structured
 273 composition that maintains clear relationships between components. Step 4 employs comprehensive
 274 self-refinement with symbolic validator feedback that performs three checks: syntax correctness, variable
 275 name faithfulness between formal and natural language representations, and logical consistency
 276 verification, ensuring premises are non-contradictory and conclusions are non-trivial. Unlike existing
 277 approaches that rely solely on compiler feedback, our validator provides targeted semantic feedback to
 278 guide iterative LLM refinement, directly addressing validation errors and unfaithful variable naming until
 279 the formalization meets all quality criteria.

280
 281
 282
 283
 284

285 3 EXPERIMENTS

286
 287

288 **Benchmarks.** We evaluate *DNA* on two benchmarks: LeanEuclidPlus and ProofNet-Hard. LeanEuclidPlus adapts LeanEuclid (Murphy et al., 2024), implementing a formal Euclidean geometry
 289 system (Avigad et al., 2009) as a domain-specific language (DSL). The scarcity of training data for
 290 LeanEuclid-style statements makes it ideal for evaluating autoformalization generalizability across
 291 DSLs. We refined the 100 UniGeo problems from LeanEuclid and added 40 hand-crafted statements
 292 with greater geometric complexity.

293
 294
 295
 296
 297
 298
 299
 300

294 Our main evaluation uses the 100 core problems for end-to-end pipeline assessment, while the additional 40 problems test generalizability of learned abstractions and decomposition-driven formalization. ProofNet-Hard comprises 19 challenging ProofNet (Azerbayev et al., 2023; Vishwakarma et al., 2024) statements requiring auxiliary helper definitions beyond standard Mathlib imports. As Table 5 shows, when helper definitions are withheld, both specialized autoformalizers—despite ProofNet training data (Wang et al., 2025; Xuejun et al., 2025)—and state-of-the-art models like GPT-5 and Claude-4-Sonnet achieve zero success rates.

301
 302
 303
 304
 305
 306
 307
 308
 309

301 **Evaluation.** Previous work uses LLM backtranslation and semantic equivalence judgment (Ying et al., 2025; Gao et al., 2025; Wang et al., 2025; Xuejun et al., 2025; Liu et al., 2025b), which lacks soundness (Liu et al., 2025a). We selected benchmarks with symbolic equivalence checkers for rigorous evaluation. For ProofNet-Hard, we use the BEq+ checker (Poiroux et al., 2025), which provides semantic equivalence verification through bidirectional entailment checking for complex logical structures. For LeanEuclidPlus, we enhanced the original E3 symbolic checker with comprehensive three-stage pre-checks to eliminate false positives and separate bidirectional verification of premises and conclusions to minimize false negatives. We report pass@1 accuracy based on successful symbolic verification.

310
 311
 312
 313
 314
 315
 316

310 **Models.** We evaluate across fine-tuned autoformalizers (Kimina (Wang et al., 2025), Mathesis (Xuejun et al., 2025)) and three major model families: GPT 4.1/5 (OpenAI, 2024),¹ Claude-4-Sonnet (Anthropic, 2024),² and Qwen3 (Yang et al., 2025),³ each with standard and reasoning variants. We sample 5 runs per problem with temperature 0.2 for non-reasoning models and 1.0 for reasoning models. Specialized autoformalizers use 1024 tokens, other models use 6144 tokens, and reasoning models have 12,288 token budgets. All models receive standardized 1-shot examples demonstrating the complete pipeline (see Appendix B for ablation).

317
 318
 319
 320

317 **Baselines.** Since *DNA* consists of two phases that can be executed individually, we design a systematic ablation study to evaluate each component’s contribution. Our baseline represents the standard autoformalization approach used in prior work, where models receive no corpus-specific abstractions and directly generate formal statements without decomposition. While all models are prompted to

321¹GPT models: gpt-4.1-mini-2025-04-14, gpt-4.1-2025-04-14, gpt-5-mini-2025-08-07, gpt-5-2025-08-07

322²Claude models: Claude-4-Sonnet-20250514, Claude-4-Sonnet-20250514-Thinking

323³Qwen3 models: Qwen3-14B, Qwen3-32B, Qwen3-235B-A22B-Instruct-2507, Qwen3-235B-A22B-Thinking-2507

provide step-by-step reasoning before producing their final answer, the critical distinction lies in whether we explicitly prompt the LLM to decompose statements into our semi-formalized hierarchical structure.

To isolate the contribution of each phase, we systematically reintroduce individual components. The “Divide” condition activates only Phase II of our framework, providing models with our 4-step statement autoformalization pipeline that includes hierarchical decomposition and self-refinement, while operating with the original DSL documentation without any corpus-specific abstractions. Conversely, the “Abstract” condition activates only Phase I, enriching the DSL with reusable definitions learned from the corpus, but requires models to generate formal statements directly without the benefit of our decomposition pipeline. Our complete “DNA” method combines both phases, providing models with both the corpus-specific abstractions from Phase I and the hierarchical decomposition capabilities from Phase II.

To establish theoretical upper bounds and validate our approach, we include oracle conditions using human expert-written corpus-specific definitions for both benchmarks. The “OracleA” condition provides these oracle abstractions to downstream models without activating our 4-step statement autoformalization pipeline, effectively testing the potential ceiling of perfect abstraction learning. The “DNOOracleA” condition combines these oracle abstractions with our complete 4-step statement autoformalization pipeline, establishing the theoretical performance ceiling for our entire approach.

4 RESULTS

Model	LeanEuclidPlus						ProofNet-Hard							
	Baseline	Divide	Abstract	DNA	OracleA	DNOOracleA	Baseline	Divide	Abstract	DNA	OracleA	DNOOracleA		
Fine-tuned Models														
Kimina Autoformalizer 7B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	8.4	1.1	8.4	7.4		
Mathesis Autoformalizer 7B	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.3	5.3	7.4	5.3		
Non-Reasoning Models														
GPT-4.1 mini	4.8	25.4	6.2	42.4	↑7.83×	3.0	48.8	↑9.17×	0.0	0.0	5.3	7.4	10.5	13.7
GPT-4.1	26.8	35.2	28.6	48.4	↑80.6%	34.2	61.2	↑1.28×	0.0	0.0	0.0	10.5	14.7	23.2
Claude 4 Sonnet	34.2	46.6	36.2	58.2	↑70.2%	36.6	60.4	↑76.6%	0.0	0.0	14.7	12.6	12.6	25.3
Qwen3 14B	1.0	5.0	4.8	9.6	↑8.60×	3.0	13.4	↑12.40×	0.0	0.0	1.1	3.2	0.0	0.0
Qwen3 32B	4.6	17.7	4.0	20.6	↑34.8×	9.2	26.4	↑4.74×	0.0	0.0	1.1	4.2	5.3	4.2
Qwen3 235B Instruct	40.8	43.6	45.4	64.4	↑57.8%	55.4	71.4	↑75.0%	0.0	0.0	1.1	9.5	2.1	16.8
Average	18.7	28.9	20.9	40.6	↑1.17×	23.6	46.9	↑1.51×	0.0	0.0	3.9	7.9	7.6	13.9
Reasoning Models														
GPT-5 mini	25.8	32.0	47.2	58.2	↑1.26×	58.0	71.2	↑1.76×	0.0	0.0	8.4	10.5	19.0	22.1
GPT-5	35.4	37.0	53.2	55.8	↑57.6%	60.2	68.4	↑93.2%	0.0	0.0	12.6	15.8	12.6	28.4
Claude 4 Sonnet Thinking	32.8	45.4	42.0	57.4	↑75.0%	50.8	64.4	↑96.3%	0.0	0.0	11.6	12.6	10.5	30.5
Qwen3 14B Thinking	25.4	27.8	33.8	38.6	↑52.0%	33.0	51.6	↑1.03×	0.0	0.0	1.1	1.1	4.2	9.5
Qwen3 32B Thinking	29.6	31.6	39.4	40.8	↑37.8%	46.8	58.2	↑96.6%	0.0	0.0	2.1	2.1	8.4	14.7
Qwen3 235B Thinking	40.2	45.4	41.6	55.6	↑38.3%	58.4	70.4	↑75.1%	0.0	0.0	5.3	5.3	9.5	19.0
Average	31.5	36.5	42.9	51.1	↑62.2%	42.9	64.0	↑1.03×	0.0	0.0	6.8	7.9	10.7	20.7

Table 5: Performance comparison across model families and experimental conditions on LeanEuclidPlus and ProofNet-Hard benchmarks. Results show pass@1 accuracy with improvements over baseline highlighted in red. DNA consistently outperforms individual components (Divide, Abstract) and often surpasses oracle conditions (OracleA).

4.1 EFFECTIVENESS OF DNA

Our analysis of Table 5 reveals several key insights about the effectiveness and generalizability of our framework across different model types and benchmark complexities.

The DNA framework demonstrates substantial performance improvements across all model categories and benchmarks. On LeanEuclidPlus, the most dramatic improvement is Qwen3-14B advancing from 1.0 to 9.6 (9.6x gain), while GPT-4.1-mini improves from 4.8 to performance levels approaching GPT-4.1. On ProofNet-Hard, baseline performance across all models is zero, making DNA’s ability to enable successful formalization particularly striking, especially considering that even GPT-5 with high reasoning effort—the same model used for Phase I language extension—fails completely in baseline scenarios without corpus-specific abstractions.

The framework proves especially beneficial for smaller non-reasoning models, demonstrating remarkable scalability advantages. The most notable improvement occurs with Qwen3-14B, which ad-

vances from merely 1.0 success rate at baseline to 9.6 with DNA augmentation, representing nearly a 10x performance gain. Similarly, GPT-4.1-mini improves from 4.8 at baseline to performance levels approaching GPT-4.1, constituting a transformative enhancement for resource-constrained applications. Reasoning models exhibit smaller relative improvements compared to non-reasoning models, which aligns with their inherent training for generating extended reasoning traces and decomposing complex problems autonomously. However, comparing baseline and “Divide” columns reveals that our decomposition still provides substantial benefits even for reasoning-capable models, as detailed in the ablation analysis presented in Appendix A.

Fine-tuned specialized autoformalizer models demonstrate complete failure to generalize across domain-specific languages, achieving zero success rates on LeanEuclidPlus due to their inability to follow instructions for generating correct LeanEuclid syntax and performing decomposition operations, failing even basic compilation syntax checks. On ProofNet-Hard, despite Mathesis Autoformalizer (Xuejun et al., 2025) being initialized from Kimina Autoformalizer (Wang et al., 2025) and Kimina being trained on ProofNet data, both models still achieve zero baseline performance. Notably, for specialized models, the “Abstract” condition outperforms “DNA” because these models cannot follow decomposition instructions effectively, and the decomposition prompts actually impede their performance. These findings conclusively demonstrate that fine-tuning specialized autoformalizer models lacks generalizability, and effective generalization requires DNA framework integration with models possessing strong instruction-following capabilities.

4.2 GENERALIZABILITY OF DNA

To assess the generalizability of both our learned abstractions and decompositional statement autoformalization approach, we evaluate Qwen3-235B-Instruct and GPT-5-mini, representing the best-performing non-reasoning and reasoning models respectively on the 100 core LeanEuclidPlus problems as presented in Table 6, to determine whether our main results extend to the 40 additional problems with greater diagrammatic complexity.

Model	Divide	Abstract	DNA
Qwen3-235B-A22B-Instruct-2507	13.5	25.0	47.0
gpt-5-mini-2025-08-07	9.5	47.5	57.5

Table 6: Ablation on 40 complex LeanEuclidPlus problems.

The generalizability results demonstrate consistent performance patterns across varying complexity levels, with DNA (Learned+4-stage) maintaining substantial advantages over both Abstract (Learned+1-stage) and Divide (Barebone+4-stage) approaches even on more challenging geometric problems, confirming the robustness and scalability of our integrated framework design.

4.3 SYNERGY BETWEEN DECOMPOSITION AND ABSTRACTION

Analysis of Table 5 reveals a consistent synergistic relationship between our two framework phases across all evaluated models and benchmarks. The complete DNA framework almost invariably outperforms both Divide (D) and Abstract (A) components when applied in isolation, demonstrating that the combination of abstraction learning and hierarchical decomposition yields performance gains that exceed the sum of their individual contributions. Remarkably, DNA frequently surpasses even the OracleA condition, which provides human expert-written abstractions, indicating that our learned abstractions coupled with systematic decomposition can match or exceed the effectiveness of carefully crafted human-designed abstraction libraries.

These empirical findings validate our theoretical framework presented in Section 2, confirming that decomposition and abstraction represent complementary facets of the autoformalization challenge rather than independent optimization targets. The synergistic effect emerges because abstraction learning enriches the target language with high-level mathematical concepts that facilitate more accurate clause-level translations during decomposition, while hierarchical decomposition exposes the precise semantic structure needed to effectively utilize these learned abstractions, creating a mutually reinforcing system that achieves optimal formalization performance.

432 5 RELATED WORK

434 **Autoformalization.** The field of autoformalization has advanced from early rule-based systems
 435 to modern LLM-based paradigms (Weng et al., 2025). While proof autoformalization has been
 436 employed in neural theorem proving (Jiang et al., 2023b) and LLM reasoning verification (Zhou
 437 et al., 2024), the lack of faithful automated evaluation metrics has led most research to focus on
 438 statement autoformalization. Recent LLM-based approaches include in-context learning (Wu et al.,
 439 2022; Azerbayev et al., 2023), supervised fine-tuning (Jiang et al., 2023a; Ying et al., 2025; Gao
 440 et al., 2025), reinforcement learning (Huang et al., 2025; Xuejun et al., 2025; Wang et al., 2025),
 441 retrieval-augmented generation (Zhang et al., 2024; Liu et al., 2025a), and post-inference sampling
 442 techniques like majority voting (Li et al., 2024a) and compiler feedback (Poiroux et al., 2025). How-
 443 ever, all aforementioned works let LLMs directly generate formal statements from the informal and
 444 treat each statement in isolation, thus overlooking two important insights: (1) statements within
 445 the same corpus often share common mathematical concepts that can be abstracted and reused; (2)
 446 mathematical statements are inherently semi-structured and can be hierarchically decomposed into
 447 simpler, more manageable clauses. Our framework leverages both insights by first formalizing com-
 448 mon concepts in the entire corpus into reusable abstractions, and then hierarchically decomposing
 449 individual statements into clauses that are formalized using the learned abstractions.

450 **Decomposition.** Decomposition, the principle of dividing complex problems into more tractable
 451 subtasks, is widely acknowledged in LLM reasoning (Huang & Chang, 2023). Theoretical works
 452 show that problems are more efficiently learned when decomposed (Wies et al., 2023), and generat-
 453 ing step-by-step solutions enables LLMs to tackle increasingly complex tasks (Li et al., 2024b). This
 454 principle is empirically validated across various diverse domains (Shwartz et al., 2020; Nye et al.,
 455 2021; Wei et al., 2023; Zelikman et al., 2023) from commonsense reasoning to code generation.
 456 In neural theorem proving, decomposition has been successfully adopted to break proof goals into
 457 simpler lemmas (Wang et al., 2023; Zhao et al., 2023; Wang et al., 2024a). However, in the context
 458 of autoformalization, decomposition remains largely unexplored. Only one recent work (Xuejun
 459 et al., 2025) uses decomposition for evaluation purposes: an LLM judge decomposes the informal
 460 statement into premises and conclusions, and then assesses the semantic equivalence between the
 461 decomposed informal statement and the predicted formalization. In contrast, our framework is the
 462 first to apply decomposition directly to the formalization process itself.

463 **Abstraction Learning.** Abstraction learning aims to automatically extract reusable knowledge from
 464 data for application to future tasks. In program synthesis, library learning focuses on extracting
 465 reusable subroutines from program corpora (Ellis et al., 2021; Wang et al., 2024b). In theorem
 466 proving, systems learn libraries of reusable lemmas to simplify future proofs (Zhou et al., 2022;
 467 Johansson et al., 2014; Kurashige et al., 2024; Singher & Itzhaky, 2021). In LLM reasoning, tool
 468 learning enables models to create reusable tools for domain-specific tasks (Yuan et al., 2024; Qu
 469 et al., 2025). Our work introduces abstraction learning to autoformalization, where the goal is
 470 to curate libraries of reusable mathematical definitions and relations. While program synthesis,
 471 theorem proving, and tool learning target executable abstractions (subroutines, sub-proofs, tools),
 472 autoformalization requires mathematical definitions that are usually non-computable, like axioms
 473 defining algebraic structures. Furthermore, unlike library learning, which requires large corpora
 474 of formal statements, our framework learns abstractions directly from natural language. To our
 475 knowledge, this is the first work to apply abstraction learning to statement autoformalization.

476 6 CONCLUSION

477 We have presented DNA, a novel framework that addresses the three fundamental challenges lim-
 478 iting autoformalization generalizability through corpus-driven abstraction learning and hierachical
 479 statement decomposition. Our approach demonstrates remarkable performance improvements, with
 480 gains ranging from $1.17\times$ to $8.60\times$ over baseline methods across diverse model architectures and
 481 domain-specific languages. Most significantly, DNA’s zero-training design enables smaller models
 482 like Qwen3-14B to achieve performance comparable to much larger baselines, while completely
 483 transforming autoformalization for challenging domains where specialized models fail entirely. The
 484 framework’s success suggests a fundamental shift toward more generalizable autoformalization ap-
 485 proaches that can adapt to new mathematical domains without extensive retraining, opening path-
 486 ways for automated mathematical reasoning across previously intractable formal language targets.

486 REFERENCES
487

488 Anthropic. Claude 4, 2024. URL <https://www.anthropic.com/news/clause-4>.

489 Jeremy Avigad, Edward Dean, and John Mumma. A formal system for Euclid’s Elements. *The*
490 *Review of Symbolic Logic*, 2(4):700–768, December 2009. ISSN 1755-0211. doi: 10.1017/
491 s1755020309990098. URL <http://dx.doi.org/10.1017/S1755020309990098>.

492 Zhangir Azerbayev, Bartosz Piotrowski, Hailey Schoelkopf, Edward W. Ayers, Dragomir Radev,
493 and Jeremy Avigad. ProofNet: Autoformalizing and formally proving undergraduate-level math-
494 ematics, 2023. URL <https://arxiv.org/abs/2302.12433>.

495 Leonardo De Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Proceedings of the*
496 *Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the*
497 *Construction and Analysis of Systems, TACAS’08/ETAPS’08*, pp. 337–340, Berlin, Heidelberg,
498 2008. Springer-Verlag. ISBN 3540787992.

499 Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt,
500 Luc Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: bootstrapping in-
501 ductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACM SIG-*
502 *PLAN International Conference on Programming Language Design and Implementation, PLDI*
503 2021, pp. 835–850, New York, NY, USA, 2021. Association for Computing Machinery. ISBN
504 9781450383912. doi: 10.1145/3453483.3454080.

505 Guoxiong Gao, Yutong Wang, Jiedong Jiang, Qi Gao, Zihan Qin, Tianyi Xu, and Bin Dong.
506 Herald: A natural language annotated Lean 4 dataset. In *The Thirteenth International Confer-
507 ence on Learning Representations*, 2025. URL <https://openreview.net/forum?id=Se6MgCtRhz>.

508 Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, François Garil-
509 lot, Stéphane Le Roux, Assia Mahboubi, Russell O’Connor, Sidi Ould Biha, Ioana Pasca,
510 Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked
511 proof of the odd order theorem. In *Proceedings of the 4th International Conference on In-
512 teractive Theorem Proving, ITP’13*, pp. 163–179, Berlin, Heidelberg, 2013. Springer-Verlag.
513 ISBN 9783642396335. doi: 10.1007/978-3-642-39634-2_14. URL https://doi.org/10.1007/978-3-642-39634-2_14.

514 Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Truong Le Hoang,
515 Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen,
516 Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta,
517 Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal
518 proof of the Kepler conjecture, 2015. URL <https://arxiv.org/abs/1501.02155>.

519 Jie Huang and Kevin Chen-Chuan Chang. Towards reasoning in large language models: A sur-
520vey. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the As-
521 sociation for Computational Linguistics: ACL 2023*, pp. 1049–1065, Toronto, Canada, July
522 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.67. URL
523 <https://aclanthology.org/2023.findings-acl.67/>.

524 Yanxing Huang, Xinling Jin, Sijie Liang, Fuwen Luo, Peng Li, and Yang Liu. FormaRL: Enhanc-
525 ing autoformalization with no labeled data. In *Second Conference on Language Modeling*, 2025.
526 URL <https://openreview.net/forum?id=Z2E11U94bq>.

527 Albert Q. Jiang, Wenda Li, and Mateja Jamnik. Multilingual mathematical autoformalization, 2023a.
528 URL <https://arxiv.org/abs/2311.03755>.

529 Albert Q. Jiang, Sean Welleck, Jin Peng Zhou, Wenda Li, Jiacheng Liu, Mateja Jamnik, Timothée
530 Lacroix, Yuhuai Wu, and Guillaume Lample. Draft, sketch, and prove: Guiding formal theorem
531 provers with informal proofs, 2023b. URL <https://arxiv.org/abs/2210.12283>.

532 Moa Johansson, Dan Rosén, Nicholas Smallbone, and Koen Claessen. Hipster: Integrating theory
533 exploration in a proof assistant. In *Intelligent Computer Mathematics*, pp. 108–122. Springer
534 International Publishing, 2014. ISBN 978-3-319-08434-3. doi: 10.1007/978-3-319-08434-3_9.

540 Cole Kurashige, Ruyi Ji, Aditya Giridharan, Mark Barbone, Daniel Noor, Shachar Itzhaky, Ranjit
 541 Jhala, and Nadia Polikarpova. CCLemma: E-graph guided lemma discovery for inductive equa-
 542 tional proofs. *Proc. ACM Program. Lang.*, 8(ICFP), August 2024. doi: 10.1145/3674653.

543

544 Zenan Li, Yifan Wu, Zhaoyu Li, Ximeng Wei, Fan Yang, Xian Zhang, and Xiaoxing Ma. Autoform-
 545 alize mathematical statements by symbolic equivalence and semantic consistency. In A. Glober-
 546 son, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in*
 547 *Neural Information Processing Systems*, volume 37, pp. 53598–53625. Curran Associates, Inc.,
 548 2024a. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/6034a661584af6c28fd97a6f23e56c0a-Paper-Conference.pdf.

549

550 Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. Chain of thought empowers transformers to
 551 solve inherently serial problems, 2024b. URL <https://arxiv.org/abs/2402.12875>.

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

555

556 Xiaoyang Liu, Kangjie Bao, Jiashuo Zhang, Yunqi Liu, Yuntian Liu, Yu Chen, Yang Jiao, and Tao
 557 Luo. ATLAS: Autoformalizing theorems through lifting, augmentation, and synthesis of data,
 558 2025b. URL <https://arxiv.org/abs/2502.05567>.

559

560 Mathlib Community. The Lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN*
 561 *International Conference on Certified Programs and Proofs (CPP '20)*, pp. 367–381. ACM, 2020.
 562 doi: 10.1145/3372885.3373824.

563

564 Leonardo de Moura and Sebastian Ullrich. The Lean 4 theorem prover and programming language.
 565 In André Platzer and Geoff Sutcliffe (eds.), *Automated Deduction – CADE 28*, pp. 625–635,
 566 Cham, 2021. Springer International Publishing. ISBN 978-3-030-79876-5.

567

568 Logan Murphy, Kaiyu Yang, Jialiang Sun, Zhaoyu Li, Anima Anandkumar, and Xujie Si. Auto-
 569 formalizing Euclidean geometry. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian
 570 Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st In-
 571 ternational Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning
 572 Research*, pp. 36847–36893. PMLR, 21–27 Jul 2024. URL <https://proceedings.mlr.press/v235/murphy24a.html>.

573

574 Tobias Nipkow, Markus Wenzel, and Lawrence C. Paulson. *Isabelle/HOL: a proof assistant for
 575 higher-order logic*. Springer-Verlag, Berlin, Heidelberg, 2002. ISBN 3540433767.

576

577 Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David
 578 Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Au-
 579 gustus Odena. Show your work: Scratchpads for intermediate computation with language models,
 580 2021. URL <https://arxiv.org/abs/2112.00114>.

581

582 OpenAI. Introducing gpt-5, 2024. URL <https://openai.com/index/introducing-gpt-5/>.

583

584 Auguste Poiroux, Gail Weiss, Viktor Kunčak, and Antoine Bosselut. Improving autoformalization
 585 using type checking, 2025. URL <https://arxiv.org/abs/2406.07222>.

586

587 Chang Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-
 588 rong Wen. Tool learning with large language models: a survey. *Front. Comput. Sci.*, 19(198343),
 589 2025. ISSN 2095-2228, 2095-2236. doi: 10.1007/s11704-024-40678-2.

590

591 Alastair David Reid, Rick Chen, Anastasios Deligiannis, David Gilday, David Hoyes, Will Keen,
 592 Ashan Pathirane, Owen Shepherd, Peter Vrabel, and Ali Mustafa Zaidi. End-to-end verification
 593 of ARM ® processors with ISA-Formal. 2016. URL <https://api.semanticscholar.org/CorpusID:45478983>.

594

595 Vered Shwartz, Peter West, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Unsupervised
 596 commonsense question answering with self-talk, 2020. URL <https://arxiv.org/abs/2004.05483>.

594 Eytan Singher and Shachar Itzhaky. Theory exploration powered by deductive synthesis. In Alexandra
 595 Silva and K. Rustan M. Leino (eds.), *Computer Aided Verification*, pp. 125–148, Cham, 2021.
 596 Springer International Publishing. ISBN 978-3-030-81688-9.

597

598 Rahul Vishwakarma, Kazumi Kasaura, Pietro Monticone, Abhijit-niser, OndaNaoto, and Ziyu Zhou.
 599 ProofNet-lean4, 2024. URL <https://github.com/rahul3613/ProofNet-lean4>.
 600 GitHub repository, accessed 2025-09-24.

601 Haiming Wang, Huajian Xin, Chuanyang Zheng, Lin Li, Zhengying Liu, Qingxing Cao, Yinya
 602 Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, Heng Liao, and Xiaodan Liang.
 603 LEGO-Prover: Neural theorem proving with growing libraries, 2023. URL <https://arxiv.org/abs/2310.00656>.

604

605 Haiming Wang, Huajian Xin, Zhengying Liu, Wenda Li, Yinya Huang, Jianqiao Lu, Zhicheng Yang,
 606 Jing Tang, Jian Yin, Zhenguo Li, and Xiaodan Liang. Proving theorems recursively, 2024a. URL
 607 <https://arxiv.org/abs/2405.14414>.

608

609 Haiming Wang, Mert Unsal, Xiaohan Lin, Mantas Baksys, Junqi Liu, Marco Dos Santos, Flood
 610 Sung, Marina Vinyes, Zhenzhe Ying, Zekai Zhu, Jianqiao Lu, Hugues de Saxe, Bolton Bailey,
 611 Chendong Song, Chenjun Xiao, Dehao Zhang, Ebony Zhang, Frederick Pu, Han Zhu, Jiawei Liu,
 612 Jonas Bayer, Julien Michel, Longhui Yu, Léo Dreyfus-Schmidt, Lewis Tunstall, Luigi Pagani,
 613 Moreira Machado, Pauline Bourigault, Ran Wang, Stanislas Polu, Thibaut Baroyer, Wen-Ding
 614 Li, Yazhe Niu, Yann Fleureau, Yangyang Hu, Zhouliang Yu, Zihan Wang, Zhilin Yang, Zhengying
 615 Liu, and Jia Li. Kimina-Prover preview: Towards large formal reasoning models with reinforce-
 616 ment learning, 2025. URL <https://arxiv.org/abs/2504.11354>.

617

618 Zora Zhiruo Wang, Graham Neubig, and Daniel Fried. TROVE: inducing verifiable and efficient
 619 toolboxes for solving programmatic tasks. In *Proceedings of the 41st International Conference
 620 on Machine Learning*, ICML’24. JMLR.org, 2024b.

621

622 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc
 623 Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models,
 624 2023. URL <https://arxiv.org/abs/2201.11903>.

625

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

628

629 Noam Wies, Yoav Levine, and Amnon Shashua. Sub-task decomposition enables learning in se-
 630 quence to sequence tasks. In *The Eleventh International Conference on Learning Representations*,
 631 2023. URL <https://openreview.net/forum?id=BrJATVZDWEH>.

632

633 Yuhuai Wu, Albert Q. Jiang, Wenda Li, Markus N. Rabe, Charles Staats, Mateja Jamnik, and
 634 Christian Szegedy. Autoformalization with large language models, 2022. URL <https://arxiv.org/abs/2205.12615>.

635

636 Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li,
 637 and Xiaodan Liang. DeepSeek-Prover: Advancing theorem proving in LLMs through large-scale
 638 synthetic data, 2024. URL <https://arxiv.org/abs/2405.14333>.

639

640 Yu Xuejun, Jianyuan Zhong, Zijin Feng, Pengyi Zhai, Roozbeh Yousefzadeh, Wei Chong Ng, Haox-
 641 ionic Liu, Ziyi Shou, Jing Xiong, Yudong Zhou, Claudia Beth Ong, Austen Jeremy Sugiarto,
 642 Yaoxi Zhang, Wai Ming Tai, Huan Cao, Dongcai Lu, Jiacheng Sun, Qiang Xu, Shen Xin, and
 643 Zhenguo Li. Mathesis: Towards formal theorem proving from natural languages, 2025. URL
 644 <https://arxiv.org/abs/2506.07047>.

645

646 An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang
 647 Gao, Chengan Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu,
 648 Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin
 649 Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Kefin Bao, Kexin Yang,
 650 Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui
 651 Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang

648 Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger
 649 Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan
 650 Qiu. Qwen3 technical report, 2025. URL <https://arxiv.org/abs/2505.09388>.

651

652 Kaiyu Yang, Jia Deng, and Danqi Chen. Generating natural language proofs with verifier-guided
 653 search, 2022. URL <https://arxiv.org/abs/2205.12443>.

654

655 Huaiyuan Ying, Zijian Wu, Yihan Geng, Zheng Yuan, Dahua Lin, and Kai Chen. Lean Workbook:
 656 A large-scale Lean problem set formalized from natural language math problems, 2025. URL
 657 <https://arxiv.org/abs/2406.03847>.

658

659 Lifan Yuan, Yangyi Chen, Xingyao Wang, Yi Fung, Hao Peng, and Heng Ji. CRAFT: Customizing
 660 LLMs by creating and retrieving from specialized toolsets. In *The Twelfth International Conference on Learning Representations*, 2024. URL <https://openreview.net/forum?id=G0vdDSt9XM>.

661

662 Eric Zelikman, Qian Huang, Gabriel Poesia, Noah D. Goodman, and Nick Haber. Parsel: Algo-
 663 rithmic reasoning with language models by composing decompositions, 2023. URL <https://arxiv.org/abs/2212.10561>.

664

665 Lan Zhang, Xin Quan, and Andre Freitas. Consistent autoformalization for constructing mathematical
 666 libraries. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4020–4033. Association for Computational Linguistics, 2024. doi:
 667 10.18653/v1/2024.emnlp-main.233. URL <http://dx.doi.org/10.18653/v1/2024.emnlp-main.233>.

668

669 Jianzhou Zhao, Santosh Nagarakatte, Milo M.K. Martin, and Steve Zdancewic. Formalizing the
 670 LLVM intermediate representation for verified program transformations. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*,
 671 POPL '12, pp. 427–440, New York, NY, USA, 2012. Association for Computing Machinery.
 672 ISBN 9781450310833. doi: 10.1145/2103656.2103709. URL <https://doi.org/10.1145/2103656.2103709>.

673

674 Xueliang Zhao, Wenda Li, and Lingpeng Kong. Decomposing the enigma: Subgoal-based demon-
 675 stration learning for formal theorem proving, 2023. URL <https://arxiv.org/abs/2305.16366>.

676

677 Jin Peng Zhou, Yuhuai Wu, Qiyang Li, and Roger Baker Grosse. REFACTOR: Learning
 678 to extract theorems from proofs, 2022. URL <https://openreview.net/forum?id=827jG3ahxL>.

679

680 Jin Peng Zhou, Charles Staats, Wenda Li, Christian Szegedy, Kilian Q. Weinberger, and Yuhuai Wu.
 681 Don't trust: Verify – grounding LLM quantitative reasoning with autoformalization, 2024. URL
 682 <https://arxiv.org/abs/2403.18120>.

683

684

685

686

687

688

689 **A ABLATION ON THE STATEMENT FORMALIZATION PHASE**

690

691 Table 7 presents a comprehensive ablation study examining the individual contributions of each
 692 component within our four-step statement formalization pipeline across multiple model architec-
 693 tures and both evaluation benchmarks. The results demonstrate that each step provides measurable
 694 and cumulative performance improvements, with decomposition into semi-formalized structure con-
 695 sistently yielding the most substantial gains across all model families, typically improving baseline
 696 performance by 15-25 percentage points for non-reasoning models and 8-15 percentage points for
 697 reasoning models.

698

699 The systematic translation step shows particularly strong benefits for maintaining semantic accuracy
 700 and reducing incorrect parameter assignments, with improvements most pronounced for smaller
 701 models that benefit from the structured approach to mapping informal concepts to formal correspon-
 702 dents. The composition step provides consistent improvements in maintaining proper quantifier
 703 scoping and logical relationships between premises and conclusions, addressing a significant source

702	703	Model	LeanEuclidPlus			
			1-Stage Direct	2-Stage Self-Refine	3-Stage Decomposition	4-Stage Decomposition
Non-Reasoning Models						
704	gpt-4.1-mini-2025-04-14	3.0	8.6	33.2	48.8	
705	gpt-4.1-2025-04-14	34.2	46.6	59.0	61.2	
706	Claude-4-Sonnet-20250514	36.6	47.8	57.0	60.4	
707	Qwen3-14B	3.0	6.8	10.2	13.4	
708	Qwen3-32B	9.2	10.2	17.0	26.4	
709	Qwen3-235B-A22B-Instruct-2507	55.4	59.2	65.6	71.4	
710	Average	23.6	29.9	40.3	46.9	
Reasoning Models						
711	gpt-5-mini-2025-08-07	58.0	60.4	70.0	71.2	
712	gpt-5-2025-08-07	60.2	61.6	68.4	68.4	
713	Claude-4-Sonnet-20250514-Thinking	50.8	54.2	60.4	64.4	
714	Qwen3-14B	33.0	37.8	50.0	51.6	
715	Qwen3-32B	46.8	55.6	57.0	58.2	
716	Qwen3-235B-A22B-2507-Thinking	58.4	59.2	67.2	70.4	
717	Average	51.2	54.8	62.2	64.0	

Table 7: Ablation study on the four-step statement formalization pipeline showing cumulative performance improvements. Each stage adds measurable benefits: 1-Stage (direct generation), 2-Stage (+ self-refinement), 3-Stage (+ decomposition), 4-Stage (+ translation). Results demonstrate the value of systematic decomposition over direct formalization.

of validation errors in direct end-to-end approaches. The self-refinement step with symbolic validator feedback contributes incremental but meaningful improvements of 3-8 percentage points across all model types, demonstrating the value of iterative correction guided by comprehensive semantic feedback rather than simple compiler error messages.

These findings validate our design choice to implement a multi-stage pipeline rather than attempting direct end-to-end formalization, as each component addresses distinct and complementary aspects of the autoformalization challenge. The ablation results also reveal that the performance gains are not merely additive but exhibit synergistic effects, particularly between decomposition and translation steps, where the structured intermediate representation enables more effective utilization of learned abstractions and reduces the cognitive load on language models during the formalization process.

B ABLATION ON THE ONE-SHOT EXAMPLE

Table 8 demonstrates the critical importance of providing structured examples for consistent task performance across different model architectures, revealing substantial performance degradation when models operate without guided examples. The results show that models without 1-shot examples exhibit significantly degraded performance, typically declining by 20-40 percentage points across both benchmarks, with increased variance in output formatting and frequent failures to adhere to domain-specific language syntactic requirements. This performance degradation is particularly pronounced for specialized autoformalizer models, which show complete task failure without proper formatting guidance, and for smaller general-purpose models that struggle to infer the expected decomposition structure from task descriptions alone.

The provision of comprehensive chain-of-thought examples serves multiple crucial functions beyond performance improvement. First, it establishes consistent formatting expectations that enable models to generate outputs compatible with our symbolic validation pipeline, reducing parsing errors and ensuring meaningful evaluation. Second, the detailed reasoning traces in the 1-shot examples demonstrate the expected progression from informal statement analysis through semi-formalized decomposition to final formal translation, providing a concrete template that models can adapt to new problem instances. Third, the examples help calibrate the level of detail required at each decomposition stage, preventing both over-simplification that loses semantic content and over-complication that introduces unnecessary complexity.

The ablation results also reveal interesting model-specific dependencies on example provision. Reasoning models show more graceful degradation without examples compared to standard models, likely due to their training on step-by-step problem solving, but still benefit substantially from structured guidance for domain-specific requirements. Conversely, fine-tuned autoformalizer models show catastrophic performance collapse without examples, highlighting their brittle nature and limited instruction-following capabilities outside their narrow training distribution. These findings

756	757	758	Model	Examples	LeanEuclidPlus																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																							
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	811	812	813	814	815	816	817	818	819	820	821	822	823	824	825	826	827	828	829	830	831	832	833	834	835	836	837	838	839	840	841	842	843	844	845	846	847	848	849	850	851	852	853	854	855	856	857	858	859	860	861	862	863	864	865	866	867	868	869	870	871	872	873	874	875	876	877	878	879	880	881	882	883	884	885	886	887	888	889	890	891	892	893	894	895	896	897	898	899	900	901	902	903	904	905	906	907	908	909	910	911	912	913	914	915	916	917	918	919	920	921	922	923	924	925	926	927	928	929	930	931	932	933	934	935	936	937	938	939	940	941	942	943	944	945	946	947	948	949	950	951	952	953	954	955	956	957	958	959	960	961	962	963	964	965	966	967	968	969	970	971	972	973	974	975	976	977	978	979	980	981	982	983	984	985	986	987	988	989	990	991	992	993	994	995	996	997	998	999	1000	1001	1002	1003	1004	1005	1006	1007	1008	1009	1010	1011	1012	1013	1014	1015	1016	1017	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	1080	1081	1082	1083	1084	1085	1086	1087	1088	1089	1090	1091	1092	1093	1094	1095	1096	1097	1098	1099	1100	1101	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	1135	1136	1137	1138	1139	1140	1141	1142	1143	1144	1145	1146	1147	1148	1149	1150	1151	1152	1153	1154	1155	1156	1157	1158	1159	1160	1161	1162	1163	1164	1165	1166	1167	1168	1169	1170	1171	1172	1173	1174	1175	1176	1177	1178	1179	1180	1181	1182	1183	1184	1185	1186	1187	1188	1189	1190	1191	1192	1193	1194	1195	1196	1197	1198	1199	1200	1201	1202	1203	1204	1205	1206	1207	1208	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	1243	1244	1245	1246	1247	1248	1249	1250	1251	1252	1253	1254	1255	1256	1257	1258	1259	1260	1261	1262	1263	1264	1265	1266	1267	1268	1269	1270	1271	1272	1273	1274	1275	1276	1277	1278	1279	1280	1281	1282	1283	1284	1285	1286	1287	1288	1289	1290	1291	1292	1293	1294	1295	1296	1297	1298	1299	1300	1301	1302	1303	1304	1305	1306	1307	1308	1309	1310	1311	1312	1313	1314	1315	1316	1317	1318	1319	1320	1321	1322	1323	1324	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	1353	1354	1355	1356	1357	1358	1359	1360	1361	1362	1363	1364	1365	1366	1367	1368	1369	1370	1371	1372	1373	1374	1375	1376	1377	1378	1379	1380	1381	1382	1383	1384	1385	1386	1387	1388	1389	1390	1391	1392	1393	1394	1395	1396	1397	1398	1399	1400	1401	1402	1403	1404	1405	1406	1407	1408	1409	1410	1411	1412	1413	1414	1415	1416	1417	1418	1419	1420	1421	1422	1423	1424	1425	1426	1427	1428	1429	1430	1431	1432	1433	1434	1435	1436	1437	1438	1439	1440	1441	1442	1443	1444	1445	1446	1447	1448	1449	1450	1451	1452	1453	1454	1455	1456	1457	1458	1459	1460	1461	1462	1463	1464	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	1496	1497	1498	1499	1500	1501	1502	1503	1504	1505	1506	1507	1508	1509	1510	1511	1512	1513	1514	1515	1516	1517	1518	1519	1520	1521	1522	1523	1524	1525	1526	1527	1528	1529	1530	1531	1532	1533	1534	1535	1536	1537	1538	1539	1540	1541	1542	1543	1544	1545	1546	1547	1548	1549	1550	1551	1552	1553	1554	1555	1556	1557	1558	1559	1560	1561	1562	1563	1564	1565	1566	1567	1568	1569	1570	1571	1572	1573	1574	1575	1576	1577	1578	1579	1580	1581	1582	1583	1584	1585	1586	1587	1588	1589	1590	1591	1592	1593	1594	1595	1596	1597	1598	1599	1600	1601	1602	1603	1604	1605	1606	1607	1608	1609	1610	1611	1612	1613	1614	1615	1616	1617	1618	1619	1620	1621	1622	1623	1624	1625	1626	1627	1628	1629	1630	1631	1632	1633	1634	1635	1636	1637	1638	1639	1640	1641	1642	1643	1644	1645	1646	1647	1648	1649	1650	1651	1652	1653	1654	1655	1656	1657	1658	1659	1660	1661	1662	1663	1664	1665	1666	1667	1668	1669	1670	1671	1672	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682	1683	1684	1685	1686	1687	1688	1689	1690	1691	1692	1693	1694	1695	1696	1697	1698	1699	1700	1701	1702	1703	1704	1705	1706	1707	1708	1709	1710	1711	1712	1713	1714	1715	1716	1717	1718	1719	1720	1721	1722	1723	1724	1725	1726	1727	1728	1729	1730	1731	1732	1733	1734	1735	1736	1737	1738	1739	1740	1741	1742	1743	1744	1745	1746	1747	1748	1749	1750	1751	1752	1753	1754	1755	1756	1757	1758	1759	1760	1761	1762	1763	1764	1765	1766	1767	1768	1769	1770	1771	1772	1773	1774	1775	1776	1777	1778	1779	1780	1781	1782	1783	1784	1785	1786	1787	1788	1789	1790	1791	1792	1793	1794	1795	1796	1797	1798	1799	1800	1801	1802	1803	1804	1805	1806	1807	1808	1809	1810	1811	1812	1813	1814	1815	1816	1817	1818	1819	1820	1821	1822	1823	1824	1825	1826	1827	1828	1829	1830	1831	1832	1833	1834	1835	1836	1837	1838	1839	1840	1841	1842	1843	1844	1845	1846	1847	1848	1849	1850	1851	1852	1853	1854	1855	1856	1857	1858	1859	1860	1861	1862	1863	1864	1865	1866	1867	1868	1869	1870	1871	1872	1873	1874	1875	1876	1877	1878	1879	1880	1881	1882	1883	1884	1885	1886	1887	1888	1889	1890	1891	1892	1893	1894	1895	1896	1897	1898	1899	1900	1901	1902	1903	1904	1905	1906	1907	1908	1909	1910	1911	1912	1913	1914	1915	1916	1917	1918	1919	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	1930	1931	1932	1933	1934	1935	1936	1937	1938	1939	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	2034	

810
811
812**LeanEuclidPlus: Extracted Concepts**

813 two distinct lines intersecting at a point in 2-dimensional Euclidean space
 814 three or more distinct points being sequentially aligned on a line in Euclidean plane
 815 two sets of points lying on opposing sides of a line in Euclidean plane
 816 two angles in Euclidean plane being supplementary
 817 two lines being parallel in 2-dimensional Euclidean space
 818 two angles in Euclidean plane being congruent
 819 a point lying between two other points on a line segment in Euclidean plane
 820 a triangle in 2-dimensional Euclidean space defined by three non-collinear points
 821 two triangles sharing a common vertex in Euclidean plane
 822 two points lying on the same side of a line in Euclidean plane
 823 two points lying on opposite sides of a line in Euclidean plane
 824 ...
 825
 826
 827
 828
 829

830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845**ProofNet-Hard: Extracted Concepts**

830 square root function mapping nonnegative real numbers to nonnegative real numbers
 831 a sequence of real numbers being convergent
 832 limit function mapping a convergent sequence of real numbers to a real number
 833 topology on a set X
 834 family of topologies on a fixed set X
 835 intersection of a family of topologies on X being a topology on X
 836 union of a family of topologies on X not necessarily being a topology on X
 837 one topology on X being finer than another topology on X (inclusion of collections)
 838 two topologies on the same set being comparable under inclusion
 839 two topologies on the same set being not comparable under inclusion
 840 basis for a topology on a set
 841 topology on \mathbb{R} generated by the basis of intervals with rational endpoints
 842 ...
 843
 844
 845

846
847
848

Figure 2: Example result from Step 1 Concept Extraction.

849
850
851
852
853
854
855
856
857
858

To illustrate the types of concepts extracted, we examine representative examples from both benchmarks shown in Figure 2. For LeanEuclidPlus, the extracted concepts include objects such as non-degenerate triangles in 2-dimensional Euclidean space, relations such as two distinct points being on a line, and various geometric functions. For ProofNetHard, which addresses more advanced mathematical domains, the extracted concepts span a broader range: objects include topologies on a set X , families of topologies on a set X , and bases for topologies on a set X ; relations include predicates such as a sequence of real numbers being convergent and the intersection of a family of topologies on X being a topology on X ; functions include the square root function mapping nonnegative real numbers to nonnegative real numbers and the limit function mapping a convergent sequence of real numbers to a real number.

859
860
861
862
863

We evaluate the quality of this extraction step using two metrics reported in Table 1, both averaged across 5 runs. The correctness metric measures what percentage of the extracted concepts are indeed non-ambiguous, well-defined, and abstract out of all extracted concepts. The recall metric quantitatively analyzes how many concepts from the oracle abstractions written by human experts are successfully identified and extracted by our system. Both metrics are evaluated manually because the output of this step consists of freeform natural language descriptions, and no reliable automated

864
 865
 866 a point lying between two other points on a line segment in Euclidean plane
 867 two points lying on the same side of a line in Euclidean plane
 868 two points lying on opposite sides of a line in Euclidean plane
 869 the measure of an angle in Euclidean plane mapping an angle to a real number in degrees
 870 a convex quadrilateral in 2-dimensional Euclidean space being a parallelogram

LeanEuclidPlus: Kept Concepts

872 two distinct lines intersecting at a point in 2-dimensional Euclidean space
 873 three or more distinct points being sequentially aligned on a line in Euclidean plane
 874 two sets of points lying on opposing sides of a line in Euclidean plane
 875 two angles in Euclidean plane being supplementary
 876 two lines being parallel in 2-dimensional Euclidean space
 877 two angles in Euclidean plane being congruent
 878 two triangles sharing a common vertex in Euclidean plane
 879 an angle at a vertex of a triangle in Euclidean plane
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910
 911
 912
 913
 914
 915
 916
 917

ProofNet-Hard: Filtered Out Concepts

square root function mapping nonnegative real numbers to nonnegative real numbers
 a sequence of real numbers being convergent
 limit function mapping a convergent sequence of real numbers to a real number
 family of topologies on a fixed set X
 intersection of a family of topologies on X being a topology on X
 union of a family of topologies on X not necessarily being a topology on X
 basis for a topology on a set X
 connected topological space (no separation into two disjoint nonempty open sets)
 Stone–Čech compactification βX of a completely regular space X
 ...

ProofNet-Hard: Kept Concepts

sequence of real numbers defined by $a_i = \sqrt{i+1} - \sqrt{i}$ for i in the natural numbers
 subbasis for a topology on a set X
 lower limit topology \mathbb{R}_l on the real line (topology with basis of half-open intervals $[a,b)$)
 K-topology \mathbb{R}_K on the real line
 topology on \mathbb{R} generated by the basis of intervals with rational endpoints
 sequence of functions $f_n: [0,1] \rightarrow \mathbb{R}$ defined by $f_n(x) = x^n$
 pointwise limit function $g: [0,1] \rightarrow \mathbb{R}$ with $g(x) = 0$ for $0 \leq x < 1$ and $g(1) = 1$
 limit point compactness (every infinite subset of X has a limit point in X)
 uniform topology on $[0,1]^\omega$ induced by the supremum metric
 ...

Figure 3: Example result from Step 2 Concept Filtering.

evaluation alternative exists for assessing the semantic correctness and completeness of concept extraction at this level.

Step 2: Concept Filtering. The second step of our abstraction learning pipeline performs concept filtering to remove duplicates and concepts already formalized in the target language, as specified by the guidelines in Appendix D.1.2. The filtering behavior and results vary significantly between our two benchmarks due to differences in the richness of their underlying formal libraries.

For LeanEuclidPlus, shown in Figure 3, the minimal nature of the target DSL results in relatively few concepts being filtered out. Representative examples include the concept of a point lying between two other points on a line segment in Euclidean space, which is filtered because it corresponds to the primitive relation between axiomatized in the target DSL. Similarly, the concept of two points lying on the same side of a line is filtered because it corresponds to the primitive relation

918 `Point.sameSide`. The relation describing two points being on opposite sides of a line is also
 919 filtered because it already exists in the target DSL as an abbreviation expressed as $\neg a.onLine$
 920 $l \wedge \neg b.onLine l \wedge \neg sameSide a b l$.
 921

922 In contrast, ProofNetHard exhibits substantially more filtering activity due to the exten-
 923 sive coverage of Lean Mathlib, as illustrated in Figure 3. Concepts such as a sequence
 924 of real numbers being convergent are filtered because they have direct formal correspon-
 925 dents in Mathlib, specifically the filter limit `Filter.Tendsto f atTop (nhds l)` de-
 926 fined in `Mathlib/Topology/Instances/Real`. The concept of a family of topologies
 927 on a set X is filtered because it is already represented as `Set (TopologicalSpace X)`
 928 or as an indexed family $\iota \rightarrow \text{TopologicalSpace } X$ with complete lattice operations in
 929 `Mathlib/Topology/Basic`. Even sophisticated mathematical concepts such as the Stone-
 930 Čech compactification of a completely regular space X are filtered because Mathlib provides com-
 931 prehensive support through `StoneCech X` along with the unit map `stoneCechUnit` and exten-
 932 sion operation `stoneCechExtend` in `Mathlib/Topology/StoneCech`.
 933

934 We evaluate this filtering step using the correctness (precision) metric reported in Table 1 for the
 935 filtering step, which measures what percentage of filtered-out concepts are indeed duplicates or
 936 already-formalized out of all filtered-out concepts, averaged across 5 runs. We deliberately focus on
 937 precision rather than recall for two important reasons. First, in the abstraction learning phase, our
 938 priority is to formalize reusable abstractions without missing potentially useful concepts, making
 939 filtering precision naturally more important than recall. Second, borderline cases exist where a natu-
 940 ral language concept does not directly correspond to exactly one formal definition but can be easily
 941 formalized by slightly adapting an existing definition. For instance, the concept of two triangles
 942 sharing a common vertex in Euclidean space has no direct corresponding formal definition, but it
 943 can be straightforwardly expressed by specifying the three vertices of each triangle. We design the
 944 filtering step to remove only those cases that are absolutely obvious duplicates or already-formalized
 945 concepts, erring on the side of caution to preserve potentially useful abstractions. The correctness
 946 metric is evaluated manually for the same reason as Step 1, since the output consists of freeform
 947 natural language and no reliable automated evaluation alternative exists.
 948

949 **Step 3: Concept Dependency Graph (CDG) Construction.** The third step of our abstraction
 950 learning pipeline constructs concept dependency graphs that analyze how each extracted concept
 951 can be formalized using existing formal definitions and other concepts, following the guidelines in
 952 Appendix D.1.3. The depth and complexity of these dependency graphs vary across benchmarks
 953 and concepts.
 954

955 For LeanEuclidPlus, Figure 4 presents an example of a depth-3 CDG where the target concept to
 956 be formalized depends on both already-available formal definitions in the target language, such as
 957 `Triangle.ofPoints` and `Line.intersectsLine`, as well as other concepts that do not yet
 958 have single direct formal correspondents and must themselves be formalized first. An illustrative
 959 example of the nuanced dependency analysis performed in this step involves the concept of a point
 960 lying on a line segment including endpoints in the Euclidean plane. This concept is superficially
 961 similar to the previously filtered concept of a point lying between two other points on a line segment
 962 in Euclidean space, which was removed because it has a direct formal correspondent in the primitive
 963 relation `between`. However, these concepts are semantically distinct: the former permits the point
 964 to be an endpoint of the line segment, while the latter does not. Consequently, the former concept is
 965 retained and can be formalized using `between`, logical disjunction, and equality. The dependency
 966 analysis also correctly identifies that the concept of the line through two distinct points in Euclidean
 967 plane is directly expressible using `Point.onLine`, inequality, and existential quantification. The
 968 concept dependency for a line intersecting two sides of a triangle in Euclidean plane is similarly
 969 constructed correctly with appropriate dependencies identified.
 970

971 For ProofNetHard, Figure 4 shows a simpler depth-2 CDG. The example concept of the lower limit
 972 topology on the real line, which is the topology with a basis of half-open intervals $[a, b)$, demon-
 973 strates correct dependency construction. The analysis correctly identifies that this concept is di-
 974 rectly expressible using the following formal correspondents: `Set (Set ℝ)`, `Ico : ℝ → ℝ → Set ℝ` repre-
 975 senting half-open intervals $[a, b)$, `TopologicalSpace.GenerateFrom : Set (Set ℝ) → TopologicalSpace ℝ`, existential
 976 quantification, conjunction, and equality to describe the generating family. This concept dependency is correctly constructed and
 977

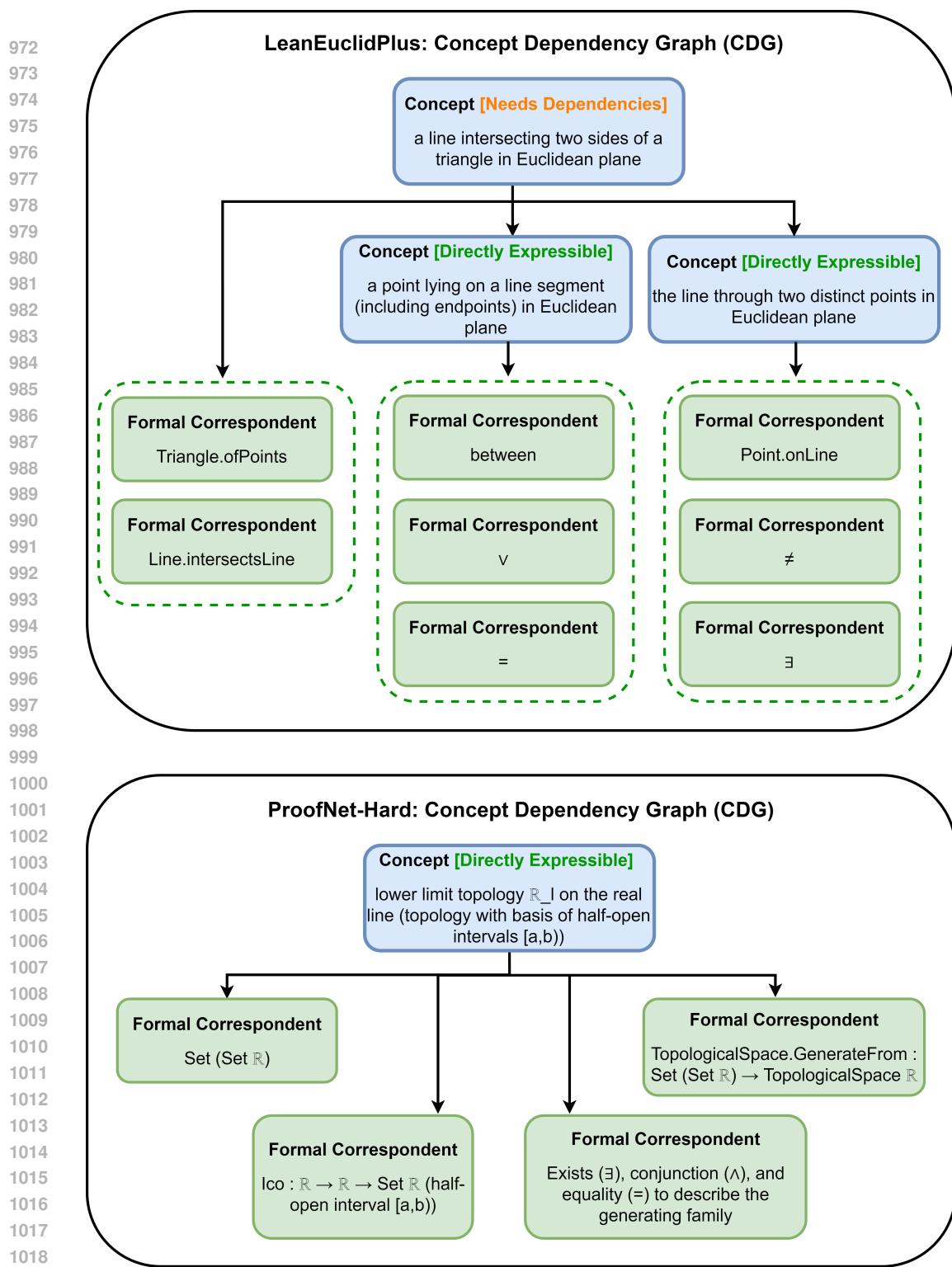


Figure 4: Example result from Step 3 CDG Construction.

precisely matches the oracle definition of the lower limit topology written by human experts, validating the accuracy of our dependency analysis.

We evaluate this CDG construction step using the correctness metric reported in Table 2, which measures what percentage of the dependency analyses for each concept are correctly constructed out of all dependency analyses, averaged across 5 runs. The correctness metric is evaluated manually for

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

LeanEuclidPlus: Formalized Concepts

```

@[simp]
def distinctFromList (x : Point) (ys : List Point) : Prop :=
  match ys with
  | [] => True
  | y :: ys => x ≠ y ∧ distinctFromList x ys

@[simp]
def pairwiseDistinct (pts : List Point) : Prop :=
  match pts with
  | [] => True
  | x :: xs => distinctFromList x xs ∧ pairwiseDistinct xs

@[simp]
def allOnLine (pts : List Point) (L : Line) : Prop :=
  match pts with
  | [] => True
  | x :: xs => x.onLine L ∧ allOnLine xs L

@[simp]
def betweennessChain (pts : List Point) : Prop :=
  match pts with
  | a :: b :: c :: rest => between a b c ∧ betweennessChain (b :: c :: rest)
  | _ => True

@[simp]
def sequentiallyAlignedOnLine (pts : List Point) (L : Line) : Prop :=
  allOnLine pts L ∧ pairwiseDistinct pts ∧ betweennessChain pts

...

```

ProofNet-Hard: Formalized Concepts

```

@[simp]
def TopologicalSpace.comparable {α : Type*} (t u : TopologicalSpace α) : Prop :=
  t ≤ u ∨ u ≤ t

@[simp]
def TopologicalSpace.notComparable {α : Type*} (t u : TopologicalSpace α) : Prop :=
  ¬ TopologicalSpace.comparable t u

@[simp]
def IsSubbasisFor {X : Type*} (S : Set (Set X)) (t : TopologicalSpace X) : Prop :=
  t = TopologicalSpace.generateFrom S

@[simp]
def lowerLimitGeneratingSets : Set (Set ℝ) :=
  {U | ∃ a b : ℝ, a < b ∧ U = Set.Ico a b}

@[simp]
def realLowerLimitTopology : TopologicalSpace ℝ :=
  TopologicalSpace.generateFrom lowerLimitGeneratingSets

...

```

Figure 5: Example result from Step 4 Concept Formalization.

1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
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

LeanEuclidPlus: Refactored Formalization

```
@[simp]
def betweennessChain (pts : List Point) : Prop :=
match pts with
| a :: b :: c :: rest => between a b c ∧ betweennessChain (b :: c :: rest)
| _ => True

@[simp]
def sequentiallyAligned (pts : List Point) : Prop :=
betweennessChain pts

...
```

ProofNet-Hard: Refactored Formalization

```
@[simp]
def TopologiesComparable {α : Type*} (t u : TopologicalSpace α) : Prop :=
t ≤ u ∨ u ≤ t

@[simp]
def IsSubbasisFor {X : Type*} (S : Set (Set X)) (t : TopologicalSpace X) : Prop :=
t = TopologicalSpace.generateFrom S

@[simp]
def LowerLimitSubbasis : Set (Set ℝ) :=
{U | ∃ a b : ℝ, a < b ∧ U = Set.Ico a b}

@[simp]
def sorgenfreyTopology : TopologicalSpace ℝ :=
TopologicalSpace.generateFrom LowerLimitSubbasis

...
```

Figure 6: Example result from Step 5 Formalization Refactoring.

the same reason as Steps 1 and 2, since the output of this step consists of freeform natural language descriptions of dependencies and formal correspondents.

Step 4: Concept Formalization. The fourth step of our abstraction learning pipeline translates the concepts identified in the CDG into actual formal definitions in the target language, following the guidelines in Appendix D.1.4. This step involves generating both the main target definitions and any necessary helper definitions to support them.

For LeanEuclidPlus, Figure 5 illustrates the formalization of the concept of three or more distinct points being sequentially aligned on a line in Euclidean plane, along with several helper definitions. The natural language description of this concept encompasses multiple sub-concepts: mutually distinct points, collinear points, and points aligned in sequential order. The language model correctly decomposes the formalization task by first formalizing each of these sub-concepts as helper definitions, then composing them to formalize the target concept. While this approach is correct and faithful to the original natural language concept, it introduces some redundancy that will be addressed in the subsequent refactoring phase.

For ProofNetHard, Figure 5 demonstrates the formalization of concepts with their helper definition dependencies. The formalization of TopologicalSpace.notComparable depends on and utilizes TopologicalSpace.Comparable as a helper definition. Similarly, realLowerLimitTopology depends on and uses lowerLimitGeneratingSets as a helper definition. Both the main concepts and their helper definitions are correctly formalized, demonstrating the system’s ability to manage dependencies during formalization.

1134 We evaluate this formalization step using the correctness metric reported in Table 2, which measures
 1135 what percentage of formal definitions are correct out of all formal definitions, with helper definitions
 1136 counted separately (essentially counting the number of function definitions), averaged across 5 runs.
 1137 Importantly, concepts do not need to be formalized by strictly following the previously analyzed
 1138 concept dependency graph; a formalization is considered correct as long as it can be interpreted
 1139 as faithful to the original natural language concept. The correctness metric is evaluated manually
 1140 despite the output being in formal language because the same natural language concept can be for-
 1141 malized in multiple ways that are all acceptable to human experts. These alternative formalizations
 1142 may be non-definitional equivalent or may not be recognized as equivalent by any existing auto-
 1143 mated checker. As discussed in Section 3, no symbolic equivalence checker is complete, and some
 1144 are not even sound, necessitating manual evaluation by human experts to assess faithfulness.
 1145

1146 **Step 5: Formalization Refactoring.** The fifth step of our abstraction learning pipeline performs
 1147 formalization refactoring to eliminate redundant definitions and extract common helper definitions,
 1148 thereby creating a more concise and general library of learned abstractions that downstream users,
 1149 whether human or language models, can better understand and employ. This refactoring process
 1150 follows the guidelines specified in Appendix D.1.5.

1151 For LeanEuclidPlus, Figure 6 demonstrates significant refactoring opportunities. As shown in the
 1152 initial formalization in Figure 5, the concept was implemented using multiple helper definitions
 1153 for mutually distinct points, collinear points, and sequential alignment. However, careful analysis
 1154 reveals that the chain of betweenness relations already guarantees that the list of points must lie
 1155 on the same line and must be mutually distinct by virtue of the between relation’s semantics.
 1156 Consequently, the two separate helper definitions for collinearity and mutual distinctness can be
 1157 eliminated as redundant. The refactored version shown in Figure 6 consolidates these concepts
 1158 by retaining only the essential betweennessChain definition, with sequentiallyAligned
 1159 serving as a more intuitive alias for the same concept.

1160 For ProofNetHard, Figure 6 shows more conservative refactoring activity. The definition
 1161 TopologicalSpace.notComparable is removed because it is simply the logical negation
 1162 concatenated with TopologicalSpace.Comparable, making a separate API unnecessary and
 1163 potentially confusing. The refactoring also includes two renamings that improve domain speci-
 1164 ficity and clarity: LowerLimitGeneratingSets is renamed to LowerLimitSubbasis
 1165 to better reflect its mathematical role, and realLowerLimitTopology is renamed to
 1166 sorgenfreyTopology using the conventional mathematical terminology. While these refac-
 1167 torings are not strictly necessary for correctness, they enhance readability and align with domain
 1168 conventions, and we therefore consider them correct refactorings.

1169 We evaluate this refactoring step using two metrics reported in Table 3. The correctness metric
 1170 measures what percentage of refactored formal definitions are correct out of all definitions that were
 1171 refactored, with helper definitions counted separately (essentially counting the number of function
 1172 definitions), averaged across 5 runs. Additionally, we quantitatively analyze the compression ratio,
 1173 which is the ratio of the number of formal definitions before refactoring to the number after refac-
 1174 toring, averaged across 5 runs. This compression ratio metric is important because it indicates how
 1175 effectively common helper functions are reused and how many unnecessary formal definitions are
 1176 eliminated. A high compression ratio facilitates better understanding and utilization of the learned
 1177 abstractions by both humans and language models, as it reduces cognitive load and clarifies the es-
 1178 sential conceptual structure. The correctness metric is evaluated manually for the same reason as
 1179 Step 4, since multiple semantically equivalent refactorings may be acceptable to human experts.
 1180

1181 **Step 6: Documentation Update.** The sixth and final step of our abstraction learning pipeline
 1182 generates clear and concise natural language documentation for the learned abstractions, abstracted
 1183 away from low-level implementation details such as internal helper definitions, to serve both human
 1184 and language model users. This step is critical because excessive definitions in the documentation
 1185 would inevitably confuse both language models as the context grows and human users attempting
 1186 to understand and use the learned abstractions. The documentation generation process follows the
 1187 guidelines specified in Appendix D.1.6.

1188 For LeanEuclidPlus, Figure 7 illustrates the selective documentation approach employed by
 1189 this step. The internal helper definition betweennessChain is not directly exposed in
 1190 the documentation; instead, its more intuitively-named alias sequentiallyAligned is pre-
 1191

Figure 7: Example result from Step 6 Documentation Update.

sented to users. The natural language description correctly captures the formal semantics of `sequentiallyAligned`, attending to subtle but important details such as the fact that the points are mutually distinct. This demonstrates the system's ability to abstract away implementation details while preserving semantic precision in user-facing documentation.

For ProofNetHard, we adopt a different approach due to the scale of the original Lean Mathlib documentation, which exceeds far beyond the context limit of any current LLMs. Instead, we skip the documentation update step and directly use the compact refactored code (comprising just 21 formal definitions) as documentation for downstream tasks in the decomposition phase.

We evaluate the documentation update step using quantitative metrics reported in Table 3. The correctness metric measures what percentage of documented formal definitions correctly convey the meaning of the definition without missing any essential details or introducing ambiguity, out of all documented definitions, averaged across 5 runs. To quantitatively validate the quality of the generated documentation, we also evaluate downstream task performance on LeanEuclidPlus using two strong models, Qwen3-235B and GPT-5, with our four-stage hierarchical decomposition pipeline. Under the hypothesis that good documentation should enable strong models to generate syntactically correct formalizations with high reliability, we report the compilation rate of the two models averaged across 5 runs. To further demonstrate the stability and robustness of the documentation produced in Step 6, we also report detailed statistics across runs: for Qwen3-235B, the maximum compilation rate is 98.2%, minimum is 96.8%, mean is 97.4%, and standard deviation is 0.47%; for GPT-5, the maximum is 100.0%, minimum is 99.0%, mean is 99.8%, and standard deviation is 0.45%. These consistently high compilation rates with low variance validate the quality and stability of the generated documentation. The correctness metric is evaluated manually for the same reason as Steps 1, 2, and 3, since the output consists of freeform natural language descriptions. The downstream task performance metrics are evaluated automatically using the same evaluation pipeline as the main experiments on LeanEuclidPlus.

1242 C.2 COMPARISON BETWEEN LEARNED AND ORACLE ABSTRACTIONS
12431244
1245 We present a qualitative comparison between the abstractions learned by our system and the ora-
1246 cle abstractions hand-crafted by human experts. Figure 8 illustrates representative examples from
1247 LeanEuclidPlus, and Figure 9 presents corresponding examples from ProofNet-Hard. Our analysis
1248 reveals three distinct categories of learned abstractions.
12491250 **Correct Overlapping Abstractions.** The first category comprises abstractions where both the
1251 learned and oracle definitions capture semantically equivalent concepts, albeit with different naming
1252 conventions or slightly different formulations.
12531254 For LeanEuclidPlus, the learned abstraction `betweennessChain` and its alias
1255 `sequentiallyAligned` correspond directly to the oracle’s `sequentiallyAlignedList`,
1256 both encoding the property that consecutive points in a list satisfy the betweenness re-
1257 lation. Similarly, `twoDistinctLinesIntersectAtPoint` mirrors the oracle’s
1258 `twoLinesIntersectAtPoint`, and the learned `parallel` and `perpendicularAt`
1259 definitions align precisely with their oracle counterparts. The learned `trianglesSimilar`
1260 captures the same geometric concept as the oracle’s `similar` abbreviation, encompassing both
1261 angle equality and proportional side lengths.
12621263 For ProofNet-Hard, the learned `sqrtSuccDiff` exactly matches the oracle’s func-
1264 tion `g`, both computing the difference between consecutive square roots. The learned
1265 `LowerLimitSubbasis`, `sorgenfreyTopology`, `KNatRecip`, `KTopologySubbasis`,
1266 and `kTopology` correspond semantically to the oracle’s `lower_limit_topology`, `K`, and
1267 `K_topology`.
12681269 These examples demonstrate that our abstraction learning phase is able to recover most of the com-
1270 mon mathematical concepts identified by human experts, validating the effectiveness of our auto-
1271 mated approach for discovering domain-specific abstractions without manual intervention.
12721273 **Correct Non-Overlapping Abstractions.** The second category consists of abstractions that are
1274 correct formalizations of the natural language concepts but do not overlap with the oracle.
12751276 For LeanEuclidPlus, the learned abstractions include `noneOnLine`, which recursively veri-
1277 fies that no point in a list lies on a given line, and `allOpposingSidesToPoint`, which
1278 checks whether all points in a list are on opposite sides of a line relative to a reference
1279 point. The learned `pairwiseAcrossOpposing` and `twoSetsOpposingSidesOnLine`
1280 provide compositional predicates for reasoning about point configurations across lines, while
1281 `lineCutsTriangleOnABandACAt` encapsulates the specific geometric configuration where
1282 a line intersects two sides of a triangle. In contrast, the oracle for LeanEuclidPlus contains dif-
1283 ferent abstractions such as `pointDistinctFromList`, `mutuallyDistinctPointsList`,
1284 `sameSideList`, and `sameSideDistinctList`, which focus on distinctness and same-side
1285 predicates over point lists.
12861287 For ProofNet-Hard, the learned abstractions include `StrictlyBoundedAboveBy`, a
1288 generic predicate for sequences bounded by a constant, `TopologiesComparable`
1289 for comparing topologies, and `NestedClosedNonempty` along with
1290 `NestedClosedNonemptyInterNonempty` for reasoning about nested sequences of closed
1291 sets. The oracle involves different abstractions such as `is_topology`, a first-principles definition
1292 of topological structure, and `countably_compact`, which directly encodes the definition of
1293 countable compactness.
12941295 These non-overlapping abstractions illustrate that concept extraction by humans and LLMs from
1296 the same natural language corpus can be complementary: each identifies valid abstractions that
1297 the other may overlook, suggesting that combining both approaches could yield richer and more
1298 comprehensive libraries of reusable abstractions.
12991300 **Correct Learned Abstractions, Wrong Oracle.** The third category, observed exclusively in
1301 ProofNet-Hard, consists of cases where the learned abstractions are correct while the oracle ab-
1302 stractions contain semantic errors. The learned `IsLimitPoint` correctly defines a limit point
1303 (accumulation point) of a set A as a point x for which every neighborhood intersects $A \setminus \{x\}$, and
1304 `LimitPointCompact` correctly states that every infinite subset has such a limit point. However,
1305

1296

1297

1298

1299

1300

1301 **LeanEuclidPlus: Correct Overlapping Abstractions**

1302

1303

```

1304 Learned Abstractions
1305 def betweennessChain (pts : List Point) : Prop := 
1306   match pts with
1307     | a :: b :: c :: rest => between a b c  $\wedge$  betweennessChain (b :: c :: rest)
1308     | _ => True
1309 
1310 def sequentiallyAligned (pts : List Point) : Prop := 
1311   betweennessChain pts
1312 
1313 def twoDistinctLinesIntersectAtPoint (L M : Line)
1314   (i : Point) : Prop := 
1315   L  $\neq$  M  $\wedge$  i.onLine L  $\wedge$  i.onLine M  $\wedge$  L.intersectsLine M
1316 
1317 def parallel (L M : Line) : Prop := 
1318   L  $\neq$  M  $\wedge$   $\neg$  L.intersectsLine M
1319 
1320 def perpendicularAt (L M : Line) (i a c : Point) : Prop := 
1321   L.intersectsLine M  $\wedge$  formRectilinearAngle a i c L M  $\wedge$   $\angle$  a:i:c =  $\perp$ 
1322 
1323 def trianglesSimilar (a b c d e f : Point) : Prop := 
1324    $\angle$  b:a:c =  $\angle$  e:d:f  $\wedge$ 
1325    $\angle$  a:b:c =  $\angle$  d:e:f  $\wedge$ 
1326    $\angle$  a:c:b =  $\angle$  d:f:e  $\wedge$ 
1327    $\frac{|\langle a-b \rangle|}{|\langle d-e \rangle|} = \frac{|\langle b-c \rangle|}{|\langle e-f \rangle|} \wedge \frac{|\langle b-c \rangle|}{|\langle e-f \rangle|} = \frac{|\langle c-a \rangle|}{|\langle f-d \rangle|}$ 
1328 
1329 ...

```

1323

1324

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

Figure 8: Comparision between Learned and Oracle Abstractions for LeanEuclidPlus.

1350

1351

1352

1353

1354

1355

1356

1357

1358

1359

1360

1361

1362

1363

1364

1365

1366

1367

1368

1369

1370

1371

1372

1373

1374

1375

1376

1377

1378

1379

1380

1381

1382

1383

1384

1385

1386

1387

1388

1389

1390

1391

1392

1393

1394

1395

1396

1397

1398

1399

1400

1401

1402

1403

ProofNet-Hard: Correct Overlapping Abstractions

Learned Abstractions

```

@[simp]
def sqrtSuccDiff (n : ℕ) : ℝ :=  

  Real.sqrt ((n : ℝ) + 1) - Real.sqrt (n : ℝ)

@[simp]
def LowerLimitSubbasis : Set (Set ℝ) :=  

  {U | ∃ a b : ℝ, a < b ∧ U = Set.Ioo a b}

@[simp]
def sorgenfreyTopology : TopologicalSpace ℝ :=  

  TopologicalSpace.generateFrom LowerLimitSubbasis

@[simp]
def KNatRecip : Set ℝ :=  

  {x | ∃ n : PNat, x = (1 : ℝ) / ((n : ℕ) : ℝ)}

@[simp]
def KTopologySubbasis : Set (Set ℝ) :=  

  {U | ∃ a b : ℝ, a < b ∧ (U = Set.Ioo a b  

  ∨ U = Set.Ioo a b \ KNatRecip)}

@[simp]
def kTopology : TopologicalSpace ℝ :=  

  TopologicalSpace.generateFrom KTopologySubbasis
...

```

Oracle Abstractions

```

@[simp]
def g (n : ℕ) : ℝ := sqrt (n + 1) - sqrt n

@[simp]
def lower_limit_topology (X : Type) [Preorder X] :=  

  TopologicalSpace.generateFrom {S : Set X |  

  ∃ a b, a < b ∧ S = Set.Ioo a b}

@[simp]
def K : Set ℝ := {r | ∃ n : ℕ, r = 1 / n}

@[simp]
def K_topology :=  

  TopologicalSpace.generateFrom  

  {S : Set ℝ | ∃ a b, a < b ∧ S = Set.Ioo a b} ∪  

  {S : Set ℝ | ∃ a b, a < b ∧ S = Set.Ioo a b \ K}

...

```

ProofNet-Hard: Correct Non-Overlapping Abstractions

Learned Abstractions

```

@[simp]
def StrictlyBoundedAboveBy {α : Type*} [LT α]  

  (s : ℕ → α) (c : α) : Prop :=  

  ∀ n, s n < c

@[simp]
def TopologiesComparable {α : Type*} :  

  (t u : TopologicalSpace α) : Prop :=  

  t ≤ u ∨ u ≤ t

@[simp]
def NestedClosedNonempty {X : Type*} :  

  [TopologicalSpace X] (C : ℕ → Set X) : Prop :=  

  Antitone C ∧ (∀ n : ℕ, IsClosed (C n))  

  ∧ (∀ n : ℕ, (C n).Nonempty)

@[simp]
def NestedClosedNonemptyInterNonempty {X : Type*} :  

  [TopologicalSpace X] : Prop :=  

  ∀ (C : ℕ → Set X),  

  Antitone C →  

  (∀ n : ℕ, IsClosed (C n)) →  

  (∀ n : ℕ, (C n).Nonempty) →  

  (∩ n, C n).Nonempty

```

Oracle Abstractions

```

@[simp]
def is_topology (X : Type*) (T : Set (Set X)) :=  

  Set.univ ∈ T ∧  

  (∀ s t, s ∈ T → t ∈ T → s ∩ t ∈ T) ∧  

  (∀ s, (∀ t ∈ s, t ∈ T) → Set.sUnion s ∈ T)

@[simp]
def countably_compact (X : Type*) [TopologicalSpace X] :=  

  ∀ U : ℕ → Set X,  

  (∀ i, IsOpen (U i)) ∧ ((Set.univ : Set X) ⊆ U i, U i) →  

  (∃ t : Finset ℕ, (Set.univ : Set X) ⊆ U i ∈ t, U i)

```

ProofNet-Hard: Correct Learned Abstractions, Wrong Oracle

Learned Abstractions

```

@[simp]
def IsLimitPoint {X : Type*} [TopologicalSpace X]  

  (A : Set X) (x : X) : Prop :=  

  ∀ U ∈ nhds x, (U ∩ (A \ {x})).Nonempty

@[simp]
def LimitPointCompact {X : Type*} [TopologicalSpace X] : Prop :=  

  ∀ A : Set X, A.Infinite → ∃ x : X, IsLimitPoint A x

```

Oracle Abstractions

```

@[simp]
def limit_point_compact (X : Type*) [TopologicalSpace X] :=  

  ∀ U : Set X, Infinite U → ∃ x ∈ U, ClusterPt x (P U)

```

Figure 9: Comparision between Learned and Oracle Abstractions for ProofNet-Hard.

1404 the oracle's `limit_point_compact` incorrectly uses `ClusterPt x (P U)`, conflating the
 1405 notions of cluster point and accumulation point.
 1406

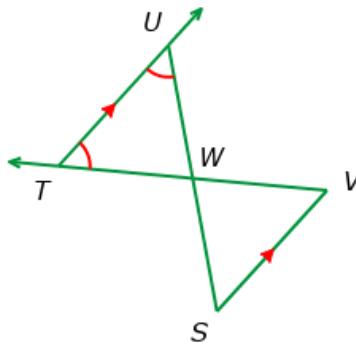
1407 In Mathlib, `ClusterPt x F` is defined as `NeBot (N x ⊓ F)`, which for a principal filter $\mathcal{P} U$
 1408 reduces to checking whether $x \in \overline{U}$. Crucially, since any point $x \in U$ trivially satisfies `ClusterPt`
 1409 $x (\mathcal{P} U)$ (every neighborhood of x contains x itself and hence meets U), the oracle's definition
 1410 degenerates to merely asserting that U is nonempty—a property that holds vacuously for any infinite
 1411 set. This renders the oracle definition trivially true in every topological space, including infinite
 1412 discrete spaces where the correct limit-point compactness property provably fails.
 1413

1414 This example demonstrates that automated abstraction learning by LLMs can sometimes produce
 1415 more reliable results than manually constructed oracles, particularly in domains where the formal-
 1416 izations contain subtle semantic distinctions that may escape human review.
 1417

C.3 HIERARCHICAL DECOMPOSITION

1418 We present a qualitative analysis demonstrating the effectiveness of our hierarchical decomposi-
 1419 tion approach (Section 2.2) for autoformalization. Using Qwen3-235B-A22B-Instruct-2507 with
 1420 learned abstractions on LeanEuclidPlus, we compare direct formalization (Abstract column in Ta-
 1421 ble 5) against our four-stage decomposition pipeline (DNA column in Table 5) on a representative
 1422 problem.
 1423

1424 **Informal Statement.** The English statement describes two triangles TUW and VSW sharing vertex
 1425 W , with lines SU and TV intersecting at W . Given that W lies between S and U , and between T
 1426 and V , along with the premises $\angle TUW = \angle UTW$ and $SV \parallel TU$, the goal is to prove
 1427 $\angle V = \angle S$.
 1428



1440 Figure 10: Diagram for Parallel-17 in LeanEuclidPlus.
 1441

1442 The following is the complete reasoning trace produced by Qwen3-235B when formalizing directly
 1443 without hierarchical decomposition:
 1444

1445 We are given an English statement involving two triangles, TUW and VSW , sharing a point W . The lines SU and TV
 1446 intersect at W , and W lies between S and U , and also between T and V . We are given that $\angle TUW =$
 1447 $\angle UTW$ (i.e., two angles in triangle TUW are congruent), and that SV is parallel to TU . We are to
 1448 prove that $\angle V = \angle S$.
 1449

1450 Let's break this down step by step.
 1451 ---
 1452

1453 **## Step 1: Identify all points, lines, and geometric entities**

1454 Points mentioned:

- 1455 - T, U, W -- form triangle TUW
- V, S, W -- form triangle VSW
- W is the shared point
- Lines: SU and TV intersect at W
- W lies between S and U \rightarrow between $S W U$
- W lies between T and V \rightarrow between $T W V$

1456 Lines to define:

- For triangle TUW : sides TU, UW, WT
- For triangle VSW : sides VS, SW, WV
- Line SU : contains S and U
- Line TV : contains T and V

1458
 1459 We are given:
 1460 1. $\angle TUW = \angle UTW$
 - $\angle TUW$ is the angle at U in triangle TUW : $\angle T:U:W$
 - $\angle UTW$ is the angle at T in triangle TUW : $\angle U:T:W$
 - So: $(\angle T:U:W) = (\angle U:T:W)$
 1461 2. $SV \parallel TU$
 - SV is the line through S and V
 - TU is the line through T and U
 - So: parallel (line SV) (line TU)
 1462 3. We are to prove: $\angle V = \angle S$
 - $\angle V$ likely means the angle at V in triangle VSW : $\angle S:V:W$ or $\angle W:V:S$?
 - $\angle S$ likely means the angle at S in triangle VSW : $\angle V:S:W$ or $\angle W:S:V$?
 1463
 1464 But note: triangle VSW has vertices V , S , W . So:
 - $\angle V$ in triangle VSW is $\angle S:V:W$
 - $\angle S$ in triangle VSW is $\angle V:S:W$
 1465 So we are to prove: $(\angle S:V:W) = (\angle V:S:W)$
 1466 But wait -- is that necessarily true? Let's see.
 1467 Alternatively, could $\angle V$ and $\angle S$ refer to angles in different triangles? But the only triangles
 are TUW and VSW .
 1468 Given that $SV \parallel TU$, and lines SU and TV intersect at W , this looks like a configuration where two lines cross
 at W , and we have a pair of parallel lines cutting across them -- possibly forming similar triangles or
 using properties of parallel lines and transversals.
 1469 But the conclusion is $\angle V = \angle S$ -- i.e., two angles at V and S , likely in triangle VSW .
 1470 Wait -- if $SV \parallel TU$, and TV and SU are transversals intersecting at W , then we might use alternate interior
 angles or corresponding angles.
 1471 But $\angle V$ and $\angle S$ are both in triangle VSW -- if $\angle S:V:W = \angle V:S:W$, that would mean
 triangle VSW is isosceles at W .
 1472 But let's reexamine.
 1473 Alternatively, could $\angle V$ mean $\angle WVS$ and $\angle S$ mean $\angle WSV$? Yes -- that's the same as
 above.
 1474 So the conclusion is: in triangle VSW , $\angle V$ equals $\angle S$ -- triangle VSW is isosceles with $|SW| = |VW|$?
 1475 But the conclusion is just $\angle V = \angle S$, i.e., angle equality.
 1476 So we are to prove: $(\angle S:V:W) = (\angle V:S:W)$
 1477 But we are given:
 - $\angle T:U:W = \angle U:T:W \rightarrow$ triangle TUW is isosceles at W ? Let's see:
 - $\angle U$ at U : $\angle T:U:W$
 - $\angle T$ at T : $\angle U:T:W$
 - If these are equal, then in triangle TUW , angles at T and U are equal \rightarrow so sides opposite are equal $\rightarrow |TW| = |UW| \rightarrow$ isosceles at W .
 1478 Also given: $SV \parallel TU$
 1479 Lines SU and TV intersect at W , with W between S and U , and between T and V .
 1480 So the configuration is: two lines (SU and TV) intersect at W , forming an "X", and two segments: TU and SV are
 parallel, with T, U on one side, S, V on the other.
 1481 This is a classic setup for similar triangles or using parallel line angle properties.
 1482 Now, since $SV \parallel TU$, and TV is a transversal (from T to V , passing through W), then:
 - $\angle UTW$ and $\angle VSW$ might be alternate interior angles? Let's see.
 1483 Wait -- transversal TV crosses parallel lines TU and SV .
 1484 But TU and SV are the parallel lines.
 1485 Transversal TV : intersects TU at T , and SV at V .
 1486 So at point T : angle between TU and TV $\rightarrow \angle U:T:W$ (since W is on TV beyond T ? Wait -- W is between T and
 V , so from T to V : $T-W-V$.
 1487 So line TV : $T \dots W \dots V$
 1488 Similarly, SU : $S \dots W \dots U$
 1489 So TU is from T to U , SV is from S to V .
 1490 Given $SV \parallel TU$.
 1491 Now, transversal TV crosses TU at T and SV at V .
 1492 So the angle at T between TU and TV (i.e., \angle between TU and TV) is $\angle U:T:W$
 1493 At V , the angle between SV and TV (i.e., between SV and VW) -- but direction matters.
 1494 Since $SV \parallel TU$, and TV is a transversal, then:

1512 - \angle UTW (at T) and \angle SVW (at V) are corresponding angles? Or alternate?
 1513
 1514 Let's define directions.
 1515 Assume standard configuration: lines TU and SV are parallel, transversal TV goes from T to V, crossing both.
 1516 Then \angle between TU and TV at T, and \angle between SV and TV at V -- if both are on the same side,
 1517 they are corresponding.
 1518 But we need to relate to the angles.
 1519 Alternatively, transversal SU: from S to U, crosses SV at S and TU at U.
 1520 So at U: angle between TU and SU $\rightarrow \angle T:U:W$ (since W is on SU beyond U? No -- W is between S and U, so
 1521 from S to U: S -- W -- U
 1522 So at U: line TU and line SU meet at U -- angle $\angle T:U:W$
 1523 At S: line SV and line SU meet at S -- angle $\angle V:S:W$
 1524 Since SV \parallel TU, and SU is a transversal, then:
 1525 - $\angle T:U:W$ and $\angle V:S:W$ are corresponding angles \rightarrow so they are equal.
 1526 So: $\angle T:U:W = \angle V:S:W$
 1527 But we are given: $\angle T:U:W = \angle U:T:W$
 1528 So: $\angle U:T:W = \angle V:S:W$
 1529 Now, look at transversal TV: from T to V, crosses TU at T and SV at V.
 1530 At T: angle between TU and TV is $\angle U:T:W$
 1531 At V: angle between SV and TV -- now, depending on direction.
 1532 Since SV \parallel TU, and TV is transversal, then $\angle U:T:W$ and $\angle S:V:W$ are alternate interior angles? Or
 1533 corresponding?
 1534 Let's suppose the lines are oriented so that TU and SV are horizontal, TU above, SV below, TV going down from
 T to V.
 1535 Then at T: angle between TU (left-right) and TV (down) -- say, lower right angle.
 1536 At V: angle between SV (left-right) and TV (up) -- upper right angle.
 1537 These are alternate interior angles \rightarrow equal.
 1538 So $\angle U:T:W = \angle S:V:W$
 1539 Therefore: $\angle S:V:W = \angle U:T:W = \angle T:U:W = \angle V:S:W$
 1540 So $\angle S:V:W = \angle V:S:W$
 1541 Thus, in triangle VSW, angles at V and S are equal $\rightarrow \angle V = \angle S$
 1542 So the conclusion is: $(\angle S:V:W) = (\angle V:S:W)$
 1543 Now we formalize.
 1544 ---
 1545 **## Step 2: Declare all variables**
 1546 Points: T, U, W, V, S
 1547 Lines:
 1548 - For triangle TUW: TU, UW, WT
 1549 - For triangle VSW: VS, SW, WV
 1550 - Line SU: contains S and U
 1551 - Line TV: contains T and V
 1552
 1553 We need to define lines:
 1554 - TU: line through T and U
 1555 - UW: line through U and W
 1556 - WT: line through W and T
 1557 - VS: line through V and S
 1558 - SW: line through S and W
 1559 - WV: line through W and V
 1560 - SU: line through S and U
 1561 - TV: line through T and V
 1562
 1563 But note: SU and TV are already mentioned.
 1564 We also need to express:
 1565 - formTriangle T U W TU UW WT
 1566 - formTriangle V S W VS SW WV
 1567 - twoDistinctLinesIntersectAtPoint SU TV W \rightarrow but SU and TV intersect at W, and are distinct
 1568 - between S W U
 1569 - between T W V
 1570 - $\angle T:U:W = \angle U:T:W \rightarrow (\angle T:U:W) = (\angle U:T:W)$
 1571 - SV \parallel TU \rightarrow parallel (line through S and V) (line through T and U)
 1572 - Conclusion: $(\angle S:V:W) = (\angle V:S:W)$

1566 But we must define the lines.
 1567
 1568 Let's assign:
 - (T U W V S : Point)
 1569 - (TU UW WT : Line) for triangle TUW
 - (VS SW WV : Line) for triangle VSW
 1570 - (SU TV : Line) for the intersecting lines
 1571 But note: SU must be the line through S and U -- but we also have SW and WU? Not necessarily.
 1572 We need to ensure that S, W, U are collinear on SU, and T, W, V are collinear on TV.
 1573
 1574 But we are given that SU and TV intersect at W, and W is between S and U, and between T and V.
 1575 So we can define:
 - SU: line through S and U -- and W is on it
 1576 - TV: line through T and V -- and W is on it
 1577 But in the triangle definitions:
 - For triangle TUW: we need lines TU, UW, WT
 1578 - But UW is part of SU? Not necessarily -- unless U, W, S are collinear, which they are -- since W is on SU
 1579 and between S and U.
 1580 So points S, W, U are collinear on line SU.
 1581 Similarly, T, W, V are collinear on line TV.
 1582 So we can define:
 - Line TU: through T and U
 1583 - Line UW: but U and W are on SU -> so UW is part of SU? But we need a line for side UW of triangle TUW.
 1584
 1585 But in `formTriangle`, we need three distinct lines: AB, BC, CA.
 1586 So for triangle TUW:
 - side TU: line TU
 1587 - side UW: line UW -- but U and W are on SU -> so line UW must be the same as line SU? Only if S, U, W are
 1588 collinear -- which they are.
 1589 But we cannot assume that line UW is the same as SU unless we define it.
 1590 But in formalization, we can let:
 - line UW be the same as line SU? But SU contains S, W, U -- so yes, line through U and W is the same as line
 1591 SU.
 1592 But in Lean, lines are extensional -- if two lines have two points in common, they are the same.
 1593 But in our formal system, we are assigning labels.
 1594 So to avoid confusion, we should define:
 - Let line SU be the line containing S and U -- and since W is between S and U, W is on SU.
 1595 - Then for triangle TUW, side UW is on line SU -- so we can use SU as the line for UW? But in `formTriangle T
 U W TU UW WT`, we need:
 1596 - T and U on TU
 - U and W on UW
 1598 - W and T on WT
 1599 So UW must be a line containing U and W -- which SU does.
 1600 Similarly, for triangle VSW:
 1601 - V and S on VS
 1602 - S and W on SW -- again, S and W are on SU? No -- S and W are on SU, but W and S are on SU.
 1603 But VS is a different line.
 1604 Wait -- triangle VSW:
 - V and S on line VS
 1605 - S and W on line SW -- but S and W are on SU -> so line SW is part of SU
 1606 - W and V on line WV
 1607 So line SW can be taken as SU? But only if we are allowed to reuse.
 1608 But in the formal system, we can have multiple names for the same line, but in `formTriangle`, we need to
 1609 assign specific lines.
 1610 So to be safe, we define:
 - Line SU: contains S, W, U
 1611 - Line TV: contains T, W, V
 - Line TU: contains T, U
 - Line VS: contains V, S
 1612 - Line WT: contains W, T -- but W and T are on TV -> so WT is part of TV
 1613 - Line WV: contains W, V -- also on TV
 1614 So for triangle TUW:
 1615 - TU: line TU
 - UW: line SU (since U and W on SU)
 1616 - WT: line TV (since W and T on TV)
 1617 Similarly, for triangle VSW:
 - VS: line VS
 - SW: line SU (S and W on SU)
 1618 - WV: line TV (W and V on TV)

1620 So we can reuse lines SU and TV in both triangles.
 1621
 1622 Now, we must ensure that the lines are distinct in each triangle.
 1623 For triangle TUW: lines TU, SU, TV -- must be distinct
 1624 For triangle VSU: lines VS, SU, TV -- must be distinct
 1625 We are not told they are distinct, but in a typical configuration they are.
 1626 But the `formTriangle` predicate requires the three lines to be distinct.
 1627 So we must assume that.
 1628 Similarly, `twoDistinctLinesIntersectAtPoint SU TV W` requires SU != TV.
 1629 Now, let's write the formal statement.
 1630 ---
 1631
 1632 **### Step 3: List all premises**
 1633 We need:
 1634 1. formTriangle T U W TU UW WT
 1635 But UW is SU, WT is TV -> so:
 1636 -> formTriangle T U W TU SU TV
 1637 But the arguments are: a b c AB BC CA
 1638 So: T, U, W, TU, SU, TV
 1639 Meaning:
 1640 - T and U on TU
 1641 - U and W on SU
 1642 - W and T on TV
 1643 And TU, SU, TV distinct.
 1644 2. formTriangle V S W VS SW WV
 1645 But SW is SU, WV is TV -> so:
 1646 -> formTriangle V S W VS SU TV
 1647 Arguments: V, S, W, VS, SU, TV
 1648 Meaning:
 1649 - V and S on VS
 1650 - S and W on SU
 1651 - W and V on TV
 1652 And VS, SU, TV distinct.
 1653 3. distinctPointsOnLine S U SU -- to ensure S and U are distinct and on SU
 1654 But `formTriangle` already requires distinctPointsOnLine for each side?
 1655 Looking back: `formTriangle` requires distinct points on each line:
 1656 - distinctPointsOnLine a b AB
 1657 - distinctPointsOnLine b c BC
 1658 - distinctPointsOnLine a c CA
 1659 So in formTriangle T U W TU SU TV:
 1660 - distinctPointsOnLine T U TU
 1661 - distinctPointsOnLine U W SU
 1662 - distinctPointsOnLine W T TV
 1663 Similarly for V S W VS SU TV:
 1664 - distinctPointsOnLine V S VS
 1665 - distinctPointsOnLine S W SU
 1666 - distinctPointsOnLine W V TV
 1667 So we get distinctness of points on lines.
 1668 But we also need that W is between S and U, and between T and V.
 1669 So:
 1670 4. between S W U
 1671 5. between T W V
 1672 6. twoDistinctLinesIntersectAtPoint SU TV W
 1673 This gives:
 1674 - SU != TV
 1675 - W on SU, W on TV
 1676 - SU intersects TV at W
 1677 But `formTriangle` already requires W on SU and W on TV?
 1678 Yes -- from distinctPointsOnLine U W SU and W T TV, etc.
 1679 But `twoDistinctLinesIntersectAtPoint` also gives that they are distinct and intersect at W -- which we
 1680 need.
 1681 But is it redundant? Possibly, but we need to ensure SU != TV, which might not be implied otherwise.
 1682 So include it.
 1683 7. $\angle T:U:W = \angle U:T:W$
 1684 -> $(\angle T:U:W) = (\angle U:T:W)$
 1685 8. SV || TU
 1686 But SV is the line VS (same line), TU is line TU
 1687 -> parallel VS TU

```

1674 Conclusion:  $\angle S:V:W = \angle V:S:W$ 
1675  $\rightarrow (\angle S:V:W) = (\angle V:S:W)$ 
1676 Note:  $\angle S:V:W$  is the angle at  $V$  in triangle  $VSW$ : points  $S, V, W$  -- so vertex  $V$ , with  $S$  and  $W$  on the
1677 sides.
1678 Similarly,  $\angle V:S:W$  is angle at  $S$ : points  $V, S, W$  -- vertex  $S$ .
1679 So we need to output:
1680 <<<  $\forall (T U W V S : \text{Point}) (TU SU TV VS : \text{Line}), [\text{premises}] \rightarrow (\angle S:V:W) = (\angle V:S:W) >>>$ 
1681 But wait -- in triangle  $VSW$ , we used lines  $VS, SU, TV$ .
1682 But  $SU$  is used for  $SW, TV$  for  $WV$ .
1683 But we also need to refer to angle at  $V$ :  $\angle S:V:W$  -- this requires points  $S, V, W$ .
1684 Similarly, angle at  $S$ :  $\angle V:S:W$  -- points  $V, S, W$ .
1685 We have all points.
1686 But in the premises, we have  $\text{formTriangle } V S W VS SU TV$  -- this ensures that:
1687 -  $V$  and  $S$  on  $VS$ 
1688 -  $S$  and  $W$  on  $SU$ 
1689 -  $W$  and  $V$  on  $TV$ 
1690 So the triangle is well-formed.
1691 Now, write the full formalization.
1692 --+
1693 ### Final Formalized Statement
1694 Variables:
1695 - Points:  $T, U, W, V, S$ 
1696 - Lines:  $TU$  (for side  $TU$ ),  $SU$  (for side  $UW$  and  $SW$ ),  $TV$  (for side  $WT$  and  $WV$ ),  $VS$  (for side  $VS$ )
1697 We use:
1698 -  $\text{formTriangle } T U W TU SU TV$ 
1699 -  $\text{formTriangle } V S W VS SU TV$ 
1700 - between  $S W U$ 
1701 - between  $T W V$ 
1702 -  $\text{twoDistinctLinesIntersectAtPoint } SU TV W$ 
1703 -  $(\angle T:U:W) = (\angle U:T:W)$ 
1704 - parallel  $VS TU$ 
1705 Conclusion:  $(\angle S:V:W) = (\angle V:S:W)$ 
1706 Note: The line  $VS$  is the same as  $SV$ ,  $TU$  is the same as  $UT$  -- direction doesn't matter for lines or parallel.
1707 So:
1708 <<<  $\forall (T U W V S : \text{Point}) (TU SU TV VS : \text{Line}), \text{formTriangle } T U W TU SU TV \land \text{formTriangle } V S W
1709 VS SU TV \land \text{between } S W U \land \text{between } T W V \land \text{twoDistinctLinesIntersectAtPoint } SU TV W
1710 \land (\angle T:U:W) = (\angle U:T:W) \land \text{parallel } VS TU \rightarrow (\angle S:V:W) = (\angle V:S:W)$ 
1711 ) >>>
1712
1713 The following is the complete reasoning trace produced by Qwen3-235B when using our four-stage
1714 hierarchical decomposition pipeline:
1715
1716 /*
1717 To semi-formalize the English Statement into a structure of premises and conclusions, we will examine the
1718 English Statement sentence by sentence, and break down each sentence into a natural language clause
1719 that is as atomic as possible.
1720 */
1721 {
1722     "premises": [
1723         /*
1724             The sentence "Two triangles  $TUW$  and  $VSW$  share a point  $W$ ." declares two triangles.
1725         */
1726         "The distinct points  $T, U$ , and  $W$  form a triangle  $\triangle T U W$ ,",
1727         "The distinct points  $V, S$ , and  $W$  form a triangle  $\triangle V S W$ ,",
1728         /*
1729             The sentence "The lines  $SU$  and  $TV$  intersect at point  $W$ ." declares two lines and their intersection.
1730         */
1731         "S and  $U$  are distinct points on line  $SU$ ,",
1732         "T and  $V$  are distinct points on line  $TV$ ,",
1733         "The distinct lines  $SU$  and  $TV$  intersect at point  $W$ ,",
1734         /*
1735             The sentence "The point  $W$  lies between  $S$  and  $U$ , and also lies between  $T$  and  $V$ ."
1736         */
1737         "Among the distinct points  $S, W, U$ , the point  $W$  is between  $S$  and  $U$ ,",
1738         "Among the distinct points  $T, W, V$ , the point  $W$  is between  $T$  and  $V$ ,",
1739         /*
1740             The sentence "Given  $\angle T U W = \angle U T W$ "
1741         */
1742         "The measure of angle  $T U W$  equals the measure of angle  $U T W$ ,",
1743     ]
1744 }

```

```

1728
1729      /*
1730      The sentence "S V || T U"
1731      */
1732      "S and V are distinct points on line SV",
1733      "T and U are distinct points on line TU",
1734      "Line SV is parallel to line TU"
1735  ],
1736  "conclusions": [
1737      "The measure of angle S V W equals the measure of angle V S W"
1738  ]
1739 */
1740      To convert the Semi-Formalized Structure into a Formalized Structure, we now formalize each clause.
1741  /*
1742  {
1743      "premises": [
1744          "formTriangle T U W TU UW WT",
1745          "formTriangle V S W VS SW WV",
1746          "distinctPointsOnLine S U SU",
1747          "distinctPointsOnLine T V TV",
1748          "twoDistinctLinesIntersectAtPoint SU TV W",
1749          "between S W U",
1750          "between T W V",
1751          "(\angle T:U:W) = (\angle U:T:W)",
1752          "distinctPointsOnLine S V VS",
1753          "distinctPointsOnLine T U TU",
1754          "parallel VS TU"
1755      ],
1756      "conclusions": [
1757          "(\angle S:V:W) = (\angle V:S:W)"
1758  ]
1759 */
1760      /*
1761      Variables: (T U W V S : Point) (TU UW WT VS SW WV SU TV : Line)
1762  */
1763
1764 <<< \forall (T U W V S : Point) (TU UW WT VS SW WV SU TV : Line),
1765     formTriangle T U W TU UW WT \land
1766     formTriangle V S W VS SW WV \land
1767     distinctPointsOnLine S U SU \land
1768     distinctPointsOnLine T V TV \land
1769     twoDistinctLinesIntersectAtPoint SU TV W \land
1770     between S W U \land
1771     between T W V \land
1772     (\angle T:U:W) = (\angle U:T:W) \land
1773     distinctPointsOnLine S V VS \land
1774     distinctPointsOnLine T U TU \land
1775     parallel VS TU
1776     ->
1777     (\angle S:V:W) = (\angle V:S:W) >>>
1778

```

Direct Formalization Without Decomposition. When prompted to formalize directly, Qwen3-235B produces the following output:

```

1764 \forall (T U W V S : Point) (TU SU TV VS : Line),
1765     formTriangle T U W TU SU TV \land formTriangle V S W VS SU TV \land
1766     between S W U \land between T W V \land
1767     twoDistinctLinesIntersectAtPoint SU TV W \land
1768     (\angle T:U:W) = (\angle U:T:W) \land parallel VS TU
1769     -> (\angle S:V:W) = (\angle V:S:W)
1770

```

This formalization is missing two critical premises: `distinctPointsOnLine S U SU` and `distinctPointsOnLine T V TV`. Without these, the statement is weaker than intended because it does not enforce that `S` and `U` are distinct points on line `SU`, nor that `T` and `V` are distinct points on line `TV`. This error type corresponds to the “Stronger/Weaker Translation” category in our error analysis (Table 4).

Notably, the model’s reasoning trace explicitly identifies these requirements. In Step 3 of its chain-of-thought, the model states: “`distinctPointsOnLine S U SU` – to ensure `S` and `U` are distinct and on `SU`” and initially considers adding this premise. However, the model then incorrectly reasons: “But `formTriangle` already requires `distinctPointsOnLine` for each side?” and lists that `formTriangle T U W TU SU TV` provides `distinctPointsOnLine U W SU`. This reasoning is flawed: in the formal language, `SU` is simply a variable name for a line—it could equivalently be called l_1 or any other identifier—and has no inherent semantic connection to the points `S` and `U`. The only way to establish that `S` and `U` are actually distinct points on this line is by explicitly stating `distinctPointsOnLine S U SU`. While `formTriangle` does constrain `U` and `W` to lie on the line named `SU`, it says nothing about whether `S` lies on this line. This

1782 illustrates a common failure mode in direct formalization: the extended free-form reasoning process
 1783 can lead to flawed logical conclusions where the model is misled by suggestive variable naming into
 1784 believing semantic relationships are already captured when they are not.

1785 **Formalization With Hierarchical Decomposition.** In contrast, our four-stage decomposition
 1786 pipeline guides the model through a structured process that prevents such errors. When processing
 1787 the sentence “The lines SU and TV intersect at point W” in Stage 2 (Semi-Formalized Structure),
 1788 the model is required to break it down into atomic clauses. This forces explicit representation of
 1789 implicit assumptions: the model produces three separate clauses—“S and U are distinct points on
 1790 line SU”, “T and V are distinct points on line TV”, and “The distinct lines SU and TV intersect
 1791 at point W”. In Stage 3 (Formalized Structure), each atomic clause is independently translated to
 1792 its formal correspondent. The clause “S and U are distinct points on line SU” simply becomes
 1793 `distinctPointsOnLine S U SU`, regardless of what other predicates might or might not im-
 1794 ply. This one-to-one translation prevents information loss and eliminates the flawed dependency
 1795 reasoning that caused the direct formalization to fail, yielding:

```
1796 \forallall (T U W V S : Point) (TU UW WT VS SW WV SU TV : Line),  

  1797   formTriangle T U W TU UW WT \land formTriangle V S W VS SW WV \land  

  1798   distinctPointsOnLine S U SU \land distinctPointsOnLine T V TV \land  

  1799   twoDistinctLinesIntersectAtPoint SU TV W \land  

  2000   between S W U \land between T W V \land  

  2001   (\angle T:U:W) = (\angle U:T:W) \land  

  2002   distinctPointsOnLine S V VS \land distinctPointsOnLine T U TU \land  

  2003   parallel VS TU  

  2004   -> (\angle S:V:W) = (\angle V:S:W)
```

2005 The decomposed output correctly includes all necessary premises. The hierarchical structure ensures
 2006 that each semantic unit from the English statement is explicitly represented as an atomic clause,
 2007 preventing information loss during the translation process.

2008 **Token Efficiency.** Beyond correctness, our decomposition approach also demonstrates improved
 2009 token efficiency. Using OpenAI’s BPE tokenizer, the direct formalization consumed 4,661 output
 2010 tokens for the free-form reasoning trace, whereas the decomposed approach used only 1,892 output
 2011 tokens, which is a 59.4% significant reduction. The structured intermediate representations serve as
 2012 a more compact and organized “working memory” compared to verbose natural language reasoning,
 2013 reducing the cost, inference time, and most importantly the long-context challenge for the model.

1812 D PROMPT TEMPLATES

1813 D.1 ABSTRACTION LEARNING

1814 D.1.1 STEP 1: CONCEPT EXTRACTION

```
1815 You are an expert in mathematics and logic with deep knowledge of all fields of mathematics. You are given  

  1816   some English mathematical statements and a list of previously extracted mathematical concepts. Your task  

  1817   is to extract ALL mathematical concepts mentioned in the current statements. {additional_specs}
```

1818 ## Mathematical Concepts
 1819 There are 3 types of mathematical concepts to extract:
 1. **Definitions of Mathematical Objects**: Point, Line, Integer, Real Number, Series, Group, Ring, etc.
 2. **Relations between Mathematical Objects**: 'a point being on a line', 'an integer being even', 'a group
 being a subgroup of another group', etc.
 3. **Functions mapping Mathematical Objects to Mathematical Objects**: 'the Euclidean distance function
 mapping two points in Euclidean space to a real number', 'the Determinant function mapping a square
 matrix to a real number', etc.

1820 ## Guidelines
 1. **Precision**: When extracting definitions, relations and functions, you MUST be as precise and detailed as
 possible. For example 'congruent' is not a precise mathematical relation! You MUST specify how many
 arguments and what type of arguments the relation or function takes like 'two integers being congruent
 modulo 3', 'two triangles being congruent', 'a list of line segments being congruent to each other', etc
 .

2. **Well-Definedness**: Please **MAKE SURE** that each mathematical object, relation, and function you
 extract is well-defined, or has a conventionally well-accepted definition. For example, 'two lines being
 on oppossing sides of a point' is not well-defined because a point can't partition a plane into sides.

3. **Abstractness**: The definitions, relations and functions you extract MUST be abstract i.e. not involving
 any particular objects, variables, names in the current statements. For example, 'line AB being parallel
 to line CD' is not abstract because it involves the names 'AB' and 'CD'.

4. **Naming Consistency**: When naming concepts, you MUST align with the previously extracted concepts if they
 refer to the same thing. For example, if previous concepts contain 'two lines being parallel in
 euclidean space' and the current statements mention 'parallel lines in euclidean space', you should use
 exactly the name 'two lines being parallel in euclidean space' for this concept extracted from the

```

1836
1837     current statements. Only add new concept names if they are truly new and not covered by previously
1838     extracted concepts.
1839
1840     4. Reasoning: Please carefully analyze the all English statements sentence by sentence, identify all
1841     potential definitions, relations and functions, and double-check that they are well-defined. Provide
1842     your detailed step-by-step reasoning BEFORE outputting your final extraction answer.
1843
1844     Output Format
1845     You MUST provide your final extraction answer i.e. a list of extracted concepts within triple angle
1846     brackets and separated by semi-colons. Each concept is a short phrase containing any necessary
1847     characters except semi-colon!!!
1848
1849     <<< concept1; concept2; ... >>>
1850
1851     Task Context:
1852     Previously Extracted Concepts:
1853     {previous_concepts}
1854
1855     Current English Statements:
1856     {english_statements}
1857
1858     D.1.2 STEP 2: CONCEPT FILTERING
1859
1860     You are an expert in mathematics, logic, programming languages, and formal verification with deep knowledge
1861     of all fields of mathematics and formalization of mathematics. You are given a list of mathematical
1862     concepts in English, and the documentation of a Domain-Specific Language (DSL) for formal mathematics. {
1863     additional_specs}
1864
1865     I want to include and formalize some of these concepts into the current formal DSL. Your task is to filter out
1866     concepts that do NOT meet the criteria for inclusion.
1867
1868     Filtering Criteria
1869     You MUST filter out concepts that satisfy ANY of the following criteria:
1870     1. Duplication of Other Concepts: The concept is a duplication of another concept. For example, the
1871     concepts 'a point being the midpoint of a line segment' and 'a point dividing a line segment into two
1872     equal parts' are duplications because they describe the exact same mathematical relationship using
1873     different wording. Please keep the one with a clearer and more conventional wording. In this case,
1874     please keep the concept 'a point being the midpoint of a line segment' and filter out the other one.
1875
1876     2. Already Defined in Current DSL: The concept has a direct corresponding formal relation/function/type
1877     already defined in the current DSL. For example, there is a concept 'two distinct points being on a line
1878     ', and in the DSL there is a formal relation 'distinctPointsOnLine' with the description: def
1879     distinctPointsOnLine (a b : Point) (L : Line) -- points a and b are distinct and on line L. The syntax
1880     is 'distinctPointsOnLine a b L'. Then the concept 'two distinct points lies on a line' should be
1881     filtered out because it is already defined in the current DSL.
1882
1883     Guidelines
1884     1. Conservative Filtering: When in doubt, KEEP the concept. Only filter out concepts that clearly
1885     violate one of the criteria.
1886
1887     2. DSL Comparison: Carefully compare each concept against the current DSL to identify redundancy or easy
1888     expressibility. Don't filter out concepts by mistake! Note that some concepts might be very similar to a
1889     formal relation/function/type in the current DSL, but are different in some subtle aspects. For example
1890     , the concept 'two points being on a line' is very similar to the formal relation 'def
1891     distinctPointsOnLine (a b : Point) (L : Line)', but the former allows the two points to be the same
1892     point, while the latter requires the two points to be distinct. For another example, the concept 'a set
1893     of points being on the same side of a line' is very similar to the formal relation 'def sameSide (a b :
1894     Point) (L : Line)', but the latter only formalizes the concept that 'two points being on the same side
1895     of a line', while the former allows the set of points to be more than two.
1896
1897     3. Reasoning: Please carefully analyze concept by concept, compare them with other concepts and the
1898     current DSL to identify redundancy, determine which concepts should be FILTERED OUT (NOT KEEPING), and
1899     provide your detailed step-by-step reasoning BEFORE outputting your final filtering answer.
1900
1901     Output Format
1902     You MUST provide your final filtering decision as a list of concepts to FILTER OUT (NOT KEEPING) within
1903     triple angle brackets and separated by semi-colons. The name of the concepts you provide should be 
1904     EXACTLY THE SAME as the name of the concepts in provided concept list!!!
1905
1906     <<< concept1; concept2; ... >>>
1907
1908     If there is no concept to filter out (KEEPING ALL CONCEPTS), please output an empty list: <<< >>>
1909
1910     Task Context:
1911     Current DSL Documentation:
1912
1913     {dsl_doc}
1914
1915     Concepts for Filtering:
1916     {concept_list}
1917
1918
1919     D.1.3 STEP 3: CDG CONSTRUCTION

```

1890 You are an expert in mathematics, logic, programming languages, and formal verification with deep knowledge of
 1891 all fields of mathematics and formalization of mathematics. You are given a list of mathematical
 1892 concepts in English, a list of previously analyzed concepts, and the documentation of a Domain-Specific
 1893 Language (DSL) for formal mathematics. (additional_specs)

1894 Your task is to analyze the dependency structure of each concept and determine how they can be formalized in
 the DSL.

1895 **## Dependency Analysis**
 1896 For each concept in the list, you need to determine:

1897 1. ****Direct Expressibility**:** Can this concept be directly expressed using existing formal types, relations,
 and functions in the current DSL? If yes, list the specific DSL elements that can express this concept.

1898 2. ****Concept Dependencies**:** If not directly expressible, what other mathematical concepts (with no direct
 correspondents in the current DSL) does this concept depend on? List all the prerequisite concepts
 1900 needed to define this concept. These dependencies should be abstract mathematical concepts, not specific
 instances.

1901 3. ****Formalization Status**:** Based on the analysis, classify the concept as:
 - 'directly_expressible': Can be directly expressed using existing DSL elements
 - 'needs_dependencies': Requires other concepts to be defined first, but ultimately formalizable
 - 'impossible': Cannot be expressed in the current DSL and would require fundamental DSL extensions

1902 **## Mathematical Concepts**
 1903 There are 3 types of mathematical concepts you can add as additional dependencies:
 1. ****Definitions of Mathematical Objects**:** Point, Line, Integer, Real Number, Series, Group, Ring, etc.
 2. ****Relations between Mathematical Objects**:** 'a point being on a line', 'an integer being even', 'a group
 1907 being a subgroup of another group', etc.
 3. ****Functions mapping Mathematical Objects to Mathematical Objects**:** 'the Euclidean distance function
 1908 mapping two points in Euclidean space to a real number', 'the Determinant function mapping a square
 matrix to a real number', etc.

1909 **## Guidelines**
 1. ****Conservative Classification**:** When in doubt between 'needs_dependencies' and 'impossible', choose '
 needs_dependencies'. Only mark as 'impossible' if the concept fundamentally cannot be expressed in the
 mathematical framework of the DSL. When in doubt between 'directly_expressible' and 'needs_dependencies
 1912 ', choose 'directly_expressible' since the DSL is designed to be expressive enough for these concepts
 1913 and you should **TRY YOUR BEST** to find directly expressible formal correspondents for these concepts.

1914 2. ****Direct Correspondents Analysis**:** Be precise about what DSL elements (types, relations, functions) can
 express each concept. Quote the exact DSL definitions and explain how they relate to the concept.

1915 3. ****Concept Dependencies Analysis**:** First of all, **TRY YOUR BEST NOT TO** introduce new concepts unless
 absolutely necessary!!! If you must introduce new concepts, note that the list of concepts you are
 1917 analyzing might have dependencies on each other. It's great if we can define one concept on top of
 another, but be careful about circular dependencies! **NO TWO CONCEPTS CAN DEPEND ON EACH OTHER!!!**

1918 4. ****Concept Dependencies Criteria**:** When adding new mathematical concepts as dependencies, you **MUST** follow
 1919 these criteria:
 a. ****Precision**:** When adding new mathematical concepts as dependencies, you **MUST** be as precise and
 detailed as possible. For example 'congruent' is not a precise mathematical relation! You **MUST**
 1921 specify how many arguments and what type of arguments the relation or function takes like 'two
 integers being congruent modulo 3', 'two triangles being congruent', 'a list of line segments being
 congruent to each other', etc.
 b. ****Well-Definedness**:** Please **MAKE SURE** that each mathematical object, relation, and function you add
 as a dependency is well-defined, or has a conventionally well-accepted definition. For example, 'two
 1924 lines being on oppossing sides of a point' is not well-defined because a point can't partition a
 plane into sides.
 c. ****Abstractness**:** The definitions, relations and functions you add as a dependency **MUST** be abstract i.e.
 1926 not involving any particular objects, variables, names in the current statements. For example, 'line
 AB being parallel to line CD' is not abstract because it involves the names 'AB' and 'CD'.

1927 4. ****Reasoning**:** Please carefully examine concept by concept, recall the definition of each concept, analyze
 1928 which other concepts are necessary to define the current concept, which of those have direct
 1929 correspondents in the current DSL, and provide your detailed step-by-step reasoning **BEFORE**
 generating the JSON output surrounded by triple angle brackets.

1930 **## Output Format**
 1931 You **MUST** provide your final analysis in JSON format within triple angle brackets. The JSON should have the
 following structure:

1932

```
```json
1933 {
1934 "concept_name_1": {
1935 "analysis": "detailed step-by-step reasoning for dependency analysis",
1936 "formal_correspondents": ["list of DSL elements that can express this concept"],
1937 "concept_dependencies": ["list of prerequisite concepts"],
1938 "status": "directly_expressible|needs_dependencies|impossible"
1939 },
1940 "concept_name_2": {
1941 ...
1942 }
1943 }
1944
1945 <<< JSON_OUTPUT_HERE >>>
1946
1947 ## Task Context:
1948 ### *Current DSL Documentation:*
```

1949 **{dsl\_doc}**

```

1944
1945 ### **Previously Analyzed Concepts:**
1946 {previous_analysis}
1947
1948 ### **Concepts for Analysis:**
1949 {concept_list}
1950
1951 D.1.4 STEP 4: CONCEPT FORMALIZATION
1952
1953 You are an expert in mathematics, logic, programming languages, and formal verification with deep knowledge of
1954 all fields of mathematics, proof assistants like Lean, and Domain-Specific Language (DSL) design. You
1955 are given a concept dependency graph (CDG) of mathematical concepts, previous extension file(s)
1956 containing the concepts you have already implemented, and the documentation of a Domain-Specific
1957 Language (DSL) for formal mathematics.
1958
1959 Your task is to extend the current DSL by implementing / formalizing the provided concepts in the CDG based on
1960 the current DSL.
1961
1962 {additional_specs}
1963
1964 ## Guidelines
1965 1. **CDG Interpretation:** In the CDG, for each concept, the field 'status' indicates if we can directly
1966 formalize this concept as a relation, function, abbreviation, alias, etc. in the current DSL. If it is 'directly_expressible', then the field 'formal_correspondents' lists the specific DSL elements that can
1967 express this concept. You **MUST** first try your best to implement the concepts using these 'formal_correspondents'. However, if they contradicts any specifications given above, You should
1968 carefully use your own knowledge and judgement to determine the most appropriate way to implement /
1969 formalize each concept based on the current DSL. If the field 'status' is 'needs_dependencies', then the
1970 field 'concept_dependencies' lists the prerequisite concepts that we need to formalize first before we
1971 can formalize this concept.
1972 2. **Faithfulness to Original Concept:** Please try your best to faithfully implement / formalize the original
1973 concept, nothing more and nothing less. For example, if the original concept is a 'a quadrilateral in
1974 2-dimensional Euclidean space', then you should only invoke the four points and four lines that form
1975 the quadrilateral, **no other elements such as the diagonal lines**!!! Please **PAY EXTRA ATTENTION** to
1976 this!!! Such subtlety will result in entirely different formalizations.
1977 3. **Code Reuse:** Please try your best to reuse the helper functions or formalized concepts from the previous
1978 extension file(s) to implement / formalize the current new concepts. Please don't reinvent the wheel!!!
1979 4. **Consistency with Current DSL:** Please ensure your extension matches the current DSL in spirit and style.
1980 For example, naming, API convention, formalization choices, etc..
1981 5. **Documentation:** Please carefully document every new type, relation, function, axiom, abbreviation, alias,
1982 notation, etc. in your implementation. Please make sure that the style and level of detail are
1983 consistent with the current DSL documentation.
1984 6. **Reasoning:** Please carefully examine concept by concept, analyze the dependency structure of each
1985 concept, determine the best way to implement / formalize each concept based on the current DSL, and
1986 provide your detailed step-by-step reasoning **BEFORE** outputting your final implementation.
1987
1988 ## Output Format
1989 You don't need to re-generate the previous extension file(s), **ONLY** output your implementation of the new
1990 concepts, which will be appended to the previous extension file(s). You **MUST** provide your final
1991 implementation within triple angle brackets.
1992 <<< your entire implementation goes here >>>
1993
1994 ## Task Context:
1995 ### **Current DSL Documentation:**
1996 {dsl_doc}
1997
1998 ### **Previous Extension File(s):**
1999 {previous_extension_files}
2000
2001 ### **Concept Dependency Graph (CDG):**
2002 {cdg_list}
2003
2004
2005 D.1.5 STEP 5: FORMALIZATION REFACTORING
2006
2007
2008 You are an expert in mathematics, logic, programming languages, and formal verification with deep knowledge of
2009 all fields of mathematics, proof assistants like Lean, and Domain-Specific Language (DSL) design. You
2010 are given the documentation of the current Domain-Specific Language (DSL) for formal mathematics and
2011 file(s) containing extensions (new definitions, relations, functions, axioms, theorems, notations, etc.)
2012 to this DSL.
2013
2014 Your task is to refactor the extension file to improve its quality, maintainability, and consistency with the
2015 DSL design principles.
2016
2017 {additional_specs}
2018
2019 ## Guidelines
2020 1. **Aggressive Refactoring:** You should be bold in refactoring the extension file:
2021 - **Eliminate Redundancy:** If multiple definitions express the exact same mathematical concept with
2022 different levels of generality, keep only the most general and well-abstracted version.

```

1998 - **Consolidate Similar Patterns**: When you find multiple similar definitions that can be unified, remove the individual ones and create a single, parameterized abstraction

1999 - **Quality Over Quantity**: It's better to have fewer, well-designed abstractions than many specific, poorly abstracted ones. You are not required to preserve every single definition from the original file. Focus on creating a clean, maintainable, and mathematically sound extension.

2000 - **Pick Best One Only**: If there are many formal relations that are similar to each other, or can substitute each other, you should either carefully choose one of them, or merge them into a single, more general and canonical definition. Multiple similar APIs can be confusing to the user, **BE CONCISE, BE CONCISE\*!!!**

2001

2002

2003

2004 2. **Preserve The Necessary**: Even though you are required to refactor aggressively, you should preserve the formal concepts are not truly not in the current DSL. It's ok if they might be easy to implement using the current DSL primitives, but this will enhance the readability and further extendability of the DSL.

2005

2006 3. **Documentation**: Please carefully update the documentation of every relation, function, axiom, abbreviation, alias, notation, etc. you refactored. Please make sure that the style and content are consistent with the current DSL documentation, NOT the comments in the extension file(s). If there are comments in the extension file(s) that are not consistent with the DSL documentation, please refactor them as well.

2007

2008 4. **Reasoning**: Please carefully analyze the extension file, identify areas for improvement according to the refactoring criteria, and provide your detailed step-by-step reasoning **BEFORE** outputting your refactored implementation.

2009

2010

2011

2012

2013

2014

2015

2016

2017

2018

2019

2020

2021

2022

2023

2024

2025

2026

2027

2028

2029

2030

2031

2032

2033

2034

2035

2036

2037

2038

2039

2040

2041

2042

2043

2044

2045

2046

2047

2048

2049

2050

2051

## Output Format  
You **MUST** provide your refactored Lean file within triple angle brackets. The output should be a complete, valid Lean file that can be appended to the DSL. Do not include any import statements - they will be handled separately.

<<< your refactored implementation goes here >>>

## Task Context:  
### Current DSL Documentation:  
{dsl\_doc}

### Extension File to Refactor:  
{extension\_file}

**D.1.6 STEP 6: DOCUMENTATION UPDATE**

You are an expert in mathematics, logic, programming languages, and formal verification with deep knowledge of all fields of mathematics, proof assistants like Lean, and Domain-Specific Language (DSL) design. You are given the documentation of the current Domain-Specific Language (DSL) for formal mathematics and extension file(s) containing new definitions, relations, functions, axioms, theorems, notations, etc. that have been added to this DSL. Your task is to generate a new DSL documentation to incorporate all the new elements from the extension file(s), while preserving the previous documentation intact.

{additional\_specs}

## Guidelines

1. **Preserve Existing Content**: Keep all existing documentation content unchanged. This is a **MUST**.

2. **Consistency with Current Doc**: Keep the style and content consistent with the current DSL documentation. For example, if the current documentation explains the syntax or gives usage examples, then you should do the same.

3. **Reasoning**: Please carefully analyze the extension file(s), identify all new elements that need to be documented, determine the best way to integrate them into the existing documentation structure, and provide your detailed step-by-step reasoning **BEFORE** outputting the new updated DSL documentation.

## Output Format  
You **MUST** provide the new DSL documentation within triple angle brackets. The output should be a complete, new version of the DSL documentation that includes everything: both elements from the current DSL documentation and new documentation for the extension file(s).

<<< your entire new DSL documentation goes here >>>

## Task Context:  
### Current DSL Documentation:  
{dsl\_doc}

### Extension File(s) to Document:  
{extension\_files}

**D.2 HIERARCHICAL DECOMPOSITION**

**D.2.1 LEANEUCLIDPLUS INSTRUCTIONS WITH LEARNED ABSTRACTION**

You are given an English Statement of a theorem from Euclidean Geometry. Note that all points and lines mentioned in the statement are distinct, unless otherwise implied by some premises.

2052 Your task is to first semi-formalize the English Statement into a json-style structure (see Guidelines #2),  
 2053 then formalize each clause in the Semi-Formalized Structure resulting in a Formalized Structure, and  
 2054 finally convert the Formalized Structure into a formal statement in Lean 4 strictly adhering to the  
 2055 following formal definitions and guidelines.

2056  $--$  Basic Geometric Sorts  $--$   
 2057 axiom Point : Type  
 2058 axiom Line : Type  
 2059 axiom Circle : Type

2060  $--$  Inductive Types for Geometric Entities  $--$   
 2061 inductive Angle | right | ofPoints (A B C : Point)  
 2062 inductive Segment | endpoints (a b : Point)  
 2063 inductive Triangle | ofPoints (a b c : Point)

2064  $--$  Notations and Macros for Geometric Entities  $--$   
 2065 " $|(a-b)|$ " means the length of the line segment (type  $\mathbb{R}$ , not type Segment) between point a and point  
 b.  
 2066 " $\angle a:b:c$ " means the degree of the angle (type  $\mathbb{R}$ , not type Angle) formed by points a, b, and c,  
 where b is the vertex of the angle and a and c are points respectively on the two sides of the angle.  
 2067 Add parentheses around the angle notation like " $(\angle a:b:c)$ " to avoid ambiguity.  
 2068 " $\perp$ " means the degree of the right angle (type  $\mathbb{R}$ , not type Angle).  
 2069 " $\triangle a:b:c$ " means the triangle (type Triangle) formed from points a, b and c.  
 2070 " $\text{Triangle.area } \triangle a:b:c$ " means the area of the triangle formed by points a, b and c. Add parentheses  
 2071 around the triangle notation like " $(\triangle a:b:c)$ " to avoid ambiguity.

2072  $--$  Relations and Axioms for Geometric Sorts  $--$   
 2073 namespace Point  
 2074 def onLine (a : Point) (L : Line) -- point a is on line L. The syntax is `a.onLine L`.  
 2075 def sameSide (a b : Point) (L : Line) -- point a and b are on the same side of line L. They both are not on  
 2076 line L, but can be the same point. The syntax is `a.sameSide b L`.  
 2077 def opposingSides (a b : Point) (L : Line) -- distinct point a and b are on opposite sides of line L. They  
 2078 both are not on line L. The syntax is `a.opposingSides b L`.  
 2079 def onCircle (a: Point) (C: Circle) -- point a is on circle C. The syntax is `a.onCircle C`.  
 2080 def insideCircle (a: Point) (C: Circle) -- point a is inside circle C. It can't be on the circle. The syntax  
 2081 is `a.insideCircle C`.  
 2082 def outsideCircle (a: Point) (C: Circle) -- point a is outside circle C. It can't be on the circle. The syntax  
 2083 is `a.outsideCircle C`.  
 2084 def isCentre (a: Point) (C: Circle) -- point a is on the unique center circle C. The syntax is `a.isCentre C`.  
 2085 def isMidpointOf (m a b : Point) -- point m is the midpoint of segment AB iff m is between a and b (hence all  
 2086 three are distinct and collinear), and  $|AM| = |MB|$ . The syntax is `m.isMidpointOf a b`.  
 2087 def isMidpointOfSegmentEndpoints (m a b : Point) (SP : Segment) -- same as `m.isMidpointOf a b` with an  
 2088 explicit segment witness. Requires `SP = Segment.endpoints a b`. The syntax is `m.  
 2089 isMidpointOfSegmentEndpoints a b SP`.  
 2090 def onExtensionBeyondB (p a b : Point) -- point p lies on the extension of segment AB beyond endpoint B iff  
 2091 'between a b p'. The syntax is `p.onExtensionBeyondB a b`.  
 2092 def onExtensionBeyondA (p a b : Point) -- point p lies on the extension of segment AB beyond endpoint A iff  
 2093 'between b a p'. The syntax is `p.onExtensionBeyondA a b`.  
 2094 end Point

2095 namespace Line  
 2096 def intersectsLine (L M : Line) -- two lines L and M intersect at some point. They can be the same line. The  
 2097 syntax is `L.intersectsLine M`.  
 2098 def intersectsCircle (L : Line) (C : Circle) -- line L and circle C intersect. The syntax is `L.  
 2099 intersectsCircle C`.  
 2099 end Line

2100 namespace Circle  
 2101 def intersectsCircle (C1 C2: Circle) -- circle C1 and C2 intersect. The syntax is `C1.intersectsCircle C2`.  
 2100 end Circle

2101 namespace Triangle  
 2102 def sideAB (a b : Point) (c : Point) : Segment -- the side AB of triangle ABC as a segment, i.e. `Segment.  
 2103 endpoints a b`. The syntax is `Triangle.sideAB a b c`.  
 2104 def sideBC (\_a : Point) (b c : Point) : Segment -- the side BC of triangle ABC as a segment, i.e. `Segment.  
 2105 endpoints b c`. The syntax is `Triangle.sideBC a b c`.

```

2106
2107 def sideCA (a : Point) (_b : Point) (c : Point) : Segment -- the side CA of triangle ABC as a segment, i.e. `Segment.endpoints c a`. The syntax is `Triangle.sideCA a b c`.
2108 end Triangle
2109
2110 -- Geometric Relations --
2111 def distinctPointsOnLine (a b : Point) (L : Line) -- points a and b are distinct and on line L. The syntax is
2112 `distinctPointsOnLine a b L`.
2113 def between (a b c : Point) -- mutually distinct points a, b and c are collinear and ordered. Point b is
2114 between point a and c. The syntax is `between a b c`.
2115 def formTriangle (a b c : Point) (AB BC CA : Line) -- mutually distinct points a, b and c form a triangle,
2116 where point a and b are on line AB, point b and c are on line BC, point a and c are on line CA. The
2117 lines AB, BC, and CA must be distinct. Note that the order and correspondence of arguments a, b, c, AB,
2118 BC, CA is strictly required i.e. a and b must be on AB, b and c must be on BC, a and c must be on CA,
2119 and they must be passed in the exact order! The syntax is "formTriangle a b c AB BC CA".
2120 def formRectilinearAngle (a b c : Point) (AB BC : Line) -- points a, b and c form a rectilinear angle, where
2121 b is the vertex of the angle and a and c are points respectively on the sides AB and BC. The sides AB
2122 and BC can be the same line, the points a and c can be the same point, but the vertex b must be distinct
2123 from a and c. The syntax is `formRectilinearAngle a b c AB BC`.
2124 def formParallelogram (a b c d : Point) (AB CD AC BD : Line) -- mutually distinct points a, b, d, and c (in
2125 clockwise/counterclockwise order i.e. ad is a diagonal) form a parallelogram, where points a and b are
2126 on line AB, points c and d are on line CD, points a and c are on line AC, and points b and d are on line
2127 BD. The lines AB, CD, AC, and BD must be distinct. Note that the order and correspondence of arguments
2128 a, b, c, d, AB, CD, AC, BD is strictly required i.e. a and b must be on AB, c and d must be on CD, a and
2129 c must be on AC, b and d must be on BD, and they must be passed in the exact order! The syntax is
2130 `formParallelogram a b c d AB CD AC BD`.
2131 def twoDistinctLinesIntersectAtPoint (L M : Line) (i : Point) -- two distinct lines L and M intersect at point
2132 i, i.e. L != M, i lies on both L and M, and `L.intersectsLine M`. The syntax is
2133 `twoDistinctLinesIntersectAtPoint L M i`.
2134 def sequentiallyAligned (pts : List Point) -- a list of points is sequentially aligned (ordered collinear
2135 chain) iff every consecutive triple (p\i, p\i+1, p\i+2) satisfies `between p\i p\i+1 p\i+2`.
2136 No extra line or global distinctness is required beyond betweenness. The syntax is `sequentiallyAligned
2137 [p\0, p\1, p\2, ...]`.
2138 def sequentiallyAlignedThreeOrMore (pts : List Point) -- like `sequentiallyAligned`, additionally requiring
2139 the list to contain at least three points. The syntax is `sequentiallyAlignedThreeOrMore [p\0, p\1, p\2,
2140 ...]`.
2141 def twoSetsOpposingSidesOnLine (xs ys : List Point) (L : Line) -- two lists of points lie on opposite sides of
2142 line L: (1) no point from either list is on L; (2) every cross-pair (x \in xs, y \in ys) is on
2143 opposing sides of L. The syntax is `twoSetsOpposingSidesOnLine [x\1, x\2, ...] [y\1, y\2, ...] L`.
2144 def supplementaryAngles (a b c d e f : Point) -- angles \angle a:b:c and \angle d:e:f are supplementary
2145 iff (\angle a:b:c) + (\angle d:e:f) = \perp + \perp. The syntax is `supplementaryAngles a b c d
2146 e f`.
2147 def parallel (L M : Line) -- two lines L and M are parallel iff they are distinct and do not intersect. The
2148 syntax is `parallel L M`.
2149 def nonCollinearPoints (a b c : Point) -- points a, b, c are non-collinear iff Triangle.area (\triangle a:b:c) != 0. The syntax is `nonCollinearPoints a b c`.
2150 def segmentFromVertexToOppositeSide (v s1 s2 p : Point) (SP : Segment) -- a segment SP connects vertex v to a
2151 point p on the opposite side s1s2, witnessed by `between s1 p s2`. Requires `SP = Segment.endpoints v p
2152 `. The syntax is `segmentFromVertexToOppositeSide v s1 s2 p SP`.
2153 def congruentSegments (a b c d : Point) -- segments AB and CD are congruent iff |AB| = |CD|. The syntax is
2154 `congruentSegments a b c d`.
2155 def perpendicularAt (L M : Line) (i a c : Point) -- lines L and M are perpendicular with right angle at
2156 witness point i using points a on L and c on M iff: (1) `L.intersectsLine M`; (2) `formRectilinearAngle
2157 a i c L M` (so a,i lie on L and i,c lie on M, with i as the vertex); (3) (\angle a:i:c) = \perp. The
2158 syntax is `perpendicularAt L M i a c`.
2159 def equalLengthRatios (a b c d e f g h : Point) -- equality of two segment-length ratios: (|AB| / |CD|) = (|EF|
2160 | / |GH|). This is a real-number equation; no non-zero-length guard is enforced. The syntax is
2161 `equalLengthRatios a b c d e f g h`.
2162 def triangleAngleSum (a b c : Point) -- for non-degenerate triangle ABC (i.e. `nonCollinearPoints a b c`), the
2163 interior angle sum is \perp + \perp: (\angle b:a:c) + (\angle a:b:c) + (\angle a:c:b) = \perp
2164 + \perp. The syntax is `triangleAngleSum a b c`.
2165 def trianglesSimilar (a b c d e f : Point) -- triangles ABC and DEF are similar iff: (1) corresponding angles
2166 are equal: \angle BAC = \angle EDF, \angle ABC = \angle DEF, \angle ACB = \angle DFE; (2)
2167 corresponding sides are proportional: |AB|/|DE| = |BC|/|EF| and |BC|/|EF| = |CA|/|FD|. The
2168 correspondence is A<->D, B<->E, C<->F. No non-degeneracy is assumed. The syntax is `trianglesSimilar a b
2169 c d e f`.
2170 def trianglesCongruent (a b c d e f : Point) -- triangles ABC and DEF are congruent iff: (1) corresponding
2171 angles are equal; (2) corresponding sides are equal: |AB|=|DE|, |BC|=|EF|, |CA|=|FD|. The correspondence
2172 is A<->D, B<->E, C<->F. The syntax is `trianglesCongruent a b c d e f`.
2173 def formConvexQuadrilateral (a b c d : Point) (AB BC CD DA : Line) -- ordered points a, b, c, d with side-
2174 lines AB, BC, CD, DA form a convex quadrilateral iff: (1) consecutive vertices lie on their side lines
2175 with distinct endpoints: `distinctPointsOnLine a b AB`, `distinctPointsOnLine b c BC`, `distinctPointsOnLine c d CD`, `distinctPointsOnLine d a DA`; (2) for each side line, the two nonincident

```

```

2160
2161 vertices lie on the same side of that line: c.sameSide d AB, d.sameSide a BC, a.sameSide b CD, b.
2162 sameSide c DA. No further distinctness, intersection, or non-collinearity is imposed. The order and
2163 correspondence are strict: AB is the side ab, BC is the side bc, CD is the side cd, DA is the side da,
2164 and they must be passed in this exact order. The syntax is `formConvexQuadrilateral a b c d AB BC CD DA
2165 `.
2166
2167 def diagonalACOfQuadrilateral (a b c d : Point) (SP : Segment) -- SP is the diagonal AC of the ordered
2168 quadruple (a, b, c, d), i.e. `SP = Segment.endpoints a c`. The syntax is `diagonalACOfQuadrilateral a b
2169 c d SP`.
2170
2171 def diagonalBDOfQuadrilateral (a b c d : Point) (SP : Segment) -- SP is the diagonal BD of the ordered
2172 quadruple (a, b, c, d), i.e. `SP = Segment.endpoints b d`. The syntax is `diagonalBDOfQuadrilateral a b
2173 c d SP`.
2174
2175 def segmentBisectsAngleAtVertex (a b c p : Point) (SP : Segment) (AB BC BP : Line) -- segment BP bisects angle
2176 ABC at vertex b, with: a,b on AB; b,c on BC; b,p on BP; interior witnessed by same-side constraints a.
2177 sameSide p BC and c.sameSide p AB; sub-angles equal `(\angle a:b:p) = (\angle p:b:c)`, and `SP =
2178 Segment.endpoints b p`. The order is strict: AB is the side ab, BC is the side bc, BP is the carrier
2179 line of the bisector bp, and they must be passed in this exact order. The syntax is `segmentBisectsAngleAtVertex a b c p SP AB BC BP`.
2180
2181 def trianglesShareSideBy
2182 (sel__1 : Point -> Point -> Segment)
2183 (sel__2 : Point -> Point -> Segment)
2184 (a b c d e f : Point) -- triangles ABC and DEF share a side witnessed by segment selectors `sel__1` and `sel
2185 __2` iff `sel__1 a b c = sel__2 d e f`. Typical selectors are `Triangle.sideAB`, `Triangle.sideBC`, or
2186 `Triangle.sideCA`. The syntax is `trianglesShareSideBy Triangle.sideAB Triangle.sideBC a b c d e f`.
2187
2188 def equilateralTriangle (a b c : Point) -- triangle ABC is equilateral iff |AB| = |BC| and |BC| = |CA|. The
2189 syntax is `equilateralTriangle a b c`.
2190
2191 def isoscelesAtA (a b c : Point) -- triangle ABC is isosceles at vertex A iff |AB| = |AC|. The syntax is `isoscelesAtA a b c`.
2192
2193 def isoscelesAtB (a b c : Point) -- triangle ABC is isosceles at vertex B iff |BA| = |BC|. The syntax is `isoscelesAtB a b c`.
2194
2195 def isoscelesAtC (a b c : Point) -- triangle ABC is isosceles at vertex C iff |CA| = |CB|. The syntax is `isoscelesAtC a b c`.
2196
2197 def lineCutsTriangleOnABandACAt (a b c p q : Point) (AB BC CA L : Line) -- line L intersects sides AB and AC
2198 of triangle ABC at points p and q respectively, with: `formTriangle a b c AB BC CA`; betweenness `between a p b` and `between c q a`;
2199 and both p and q on L. The order and correspondence are strict: AB is the side ab, BC is the side bc, CA is the side ca, and L is the cutting line; they must be passed in
2200 this exact order. The syntax is `lineCutsTriangleOnABandACAt a b c p q AB BC CA L`.
2201
2202 def lineCutsTriangleOnABandBCAt (a b c p q : Point) (AB BC CA L : Line) -- line L intersects sides AB and BC
2203 at points p and q respectively, with: `formTriangle a b c AB BC CA`; betweenness `between a p b` and `between b q c`;
2204 and both p and q on L. The order and correspondence are strict as above. The syntax is `lineCutsTriangleOnABandBCAt a b c p q AB BC CA L`.
2205
2206 def lineCutsTriangleOnACandBCAt (a b c p q : Point) (AB BC CA L : Line) -- line L intersects sides AC and BC
2207 at points p and q respectively, with: `formTriangle a b c AB BC CA`; betweenness `between c p a` and `between b q c`;
2208 and both p and q on L. The order and correspondence are strict as above. The syntax is `lineCutsTriangleOnACandBCAt a b c p q AB BC CA L`.
2209
2210 def lineCutsABandACParallelBC (a b c p q : Point) (AB BC CA L : Line) -- line L cuts sides AB and AC at p and
2211 q and is parallel to side BC; requires `lineCutsTriangleOnABandACAt a b c p q AB BC CA L` and `parallel
2212 L BC`. The syntax is `lineCutsABandACParallelBC a b c p q AB BC CA L`.
2213
2214 def lineCutsABandBCParallelCA (a b c p q : Point) (AB BC CA L : Line) -- line L cuts sides AB and BC at p and
2215 q and is parallel to side CA; requires `lineCutsTriangleOnABandBCAt a b c p q AB BC CA L` and `parallel
2216 L CA`. The syntax is `lineCutsABandBCParallelCA a b c p q AB BC CA L`.
2217
2218 def lineCutsACandBCParallelAB (a b c p q : Point) (AB BC CA L : Line) -- line L cuts sides AC and BC at p and
2219 q and is parallel to side AB; requires `lineCutsTriangleOnACandBCAt a b c p q AB BC CA L` and `parallel
2220 L AB`. The syntax is `lineCutsACandBCParallelAB a b c p q AB BC CA L`.
2221
2222 -- Guidelines --
2223 1. Formalized Statement Format: Your formalized statement must be of the form <<< \forall (...) , P_1 \land
2224 P_2 ... \land P_n -> Q_1 \land Q_2 ... \land Q_m >>> where where each P_i and Q_i is built from
2225 the above building blocks using conjunction (\land) disjunction (\lor) and negation (\neg). All
2226 variable declarations must be placed in parentheses after the universal quantifier. You shouldn't
2227 declare variables or use quantifiers in any other places! For different types of variables, you should
2228 put them in different parentheses. For example, if the English statement contains "The points A, B are
2229 on line AB", then you should declare (A B : Point) (AB : Line) in the formalized statement.
2230
2231 2. Semi-Formalized Structure: For the Semi-Formalized Structure, you must output a json5-style structure (you
2232 can interleave comments in between data) with 2 fields "premises" and "conclusions", and the value of
2233 each filed is a list of clauses.
2234
2235 {
2236 "premises": [
2237 English clause 1,
2238 English clause 2,
2239 ...
2240],
2241 "conclusions": [
2242 English clause 1,
2243 English clause 2,
2244 ...
2245]
2246 }

```

2214     }

2215     

2216     Each clause is an English sentence representing a proposition. The relationship between premises and  
2217     conclusions is implication. The premises are the antecedent and the conclusions are the consequent. The  
2218     relationship between each clause is conjunction. You should try to make each clause as atomic as  
2219     possible i.e. within a clause, there shouldn't be any logical operators. If there has to be some, you  
2220     should use as few logical operators within a clause as possible, and use multiple clauses instead to  
2221     express the same meaning.

2222     3. Formalized Structure: The structure is exactly the same as the Semi-Formalized Structure, but you will  
2223     formalize each English clauses in premises and conclusions into Lean expressions, to get the Formalized  
2224     Structure.

2225     {  
2226         "premises": [  
2227             Formalized clause 1,  
2228             Formalized clause 2,  
2229             ...  
2230         ],  
2231         "conclusions": [  
2232             Formalized clause 1,  
2233             Formalized clause 2,  
2234             ...  
2235         ]  
2236     }  
2237  
2238     For the formalization of each clause, you should first try to find if there is a direct corresponding formal  
2239     relation provided. For example, the English sentence "distinct points a is between b and c" has a  
2240     potential directly corresponding formal relation 'between'. You should then double check the detailed  
2241     description of the formal relation to see if the details match the actual English clause, and pass the  
2242     arguments in the correct order specified in the description: 'between b a c'. If no direct formalization  
2243     is available, or the relation description does not match, you should formalize it equivalently using  
2244     available operators, constants, and other formal relations. For example, "the points A, B, C form an  
2245     inscribed triangle in circle \alpha" doesn't have any direct formalization, but it's equivalent to "  
2246     the points A, B, C form a triangle, and they are all on circle \alpha", which can be formalized as  
2247     `formTriangle A B C AB BC CA \land A.onCircle \alpha \land B.onCircle \alpha \land C.onCircle  
2248     \alpha`.

2249     4. Implication: There can be only a single implication in the formula; either side of the implication must be  
2250     a conjunction of formulae.

2251     5. Variable Naming: You should always use the EXACT SAME variable name in the formalized statement as the one  
2252     in the English statement. For example, if the English statement contains "Point A, B is on line AB",  
2253     then you should use the variable names (A B : Point) (AB : Line) in the formalized statement. \*\*You must  
2254     not add any prefix or suffix to the variable names like "point\_A" or "AB\_line"!!!!\*\* Somtimes the  
2255     English statement might first refer to a line as "AB", but later use "BA" for the same line, in which  
2256     case you should use the first occurring name "AB" for that same line consistently in your formalized  
2257     statement.

2258     6. Numeric Values Restrictions: Denote 90-degree angle by \perp, 180-degree angle by \perp + \perp, etc.  
2259     Also, when referring to segments, we always mean its length (i.e. |(a-b)|).

2260     7. Quantified Variables: Your quantified variables must be limited to primitive geometric types: points, lines  
2261     , and circles. ALL bound variables that you declared must be mentioned in some clauses later.

2262     8. Intermediate Variables: You should never define an intermediate variable inside the proposition. For  
2263     example, "let \alpha := (something);" is not allowed.

2264     9. Numeric Operators: You should only uses addition (+), subtraction (-), multiplication (\*), and division  
2265     (/). Avoid using other mathematical operators such as exponentiation.

2266     10. Equality Relation: You can use the equality relation (=) and non-equality relation (!=) to compare points,  
2267     lines, circles, angles, and line segments. However, please \*\*avoid\*\* using the equality relation to  
2268     compare expressions and numeric values as it's not supported by the current formal language  
2269     implementation. For example, you should write "|(a-b)| = |(c-d)|" instead of "|(a-b)| / |(c-d)| = 1".

2270     11. Angle Notation: When you see a short-hand angle notation like "\angle X", this means that X is the  
2271     vertex of the angle. You should always expand it to the full angle notation "\angle A X B" (formally,  
2272     \angle A:X:B) where A and B are points respectively on the two sides of the angle.

2273     12. Syntax Tip 1: You should only use the provided relations and axioms. Please examine the provided axioms  
2274     and relations carefully to find the best way to express your proposition. Don't create or use new ones  
2275     that are not provided above! When you see an error message that contains "unknown identifier", it means  
2276     that you might have used some relations that are not provided in the guidelines. Please double-check  
2277     this, or your formalized statement won't compile!

2278     13. Syntax Tip 2: You must declare all Points, Lines, and Circles that you will use in the formal statement.  
2279     You shouldn't declare any extra Points, Lines, or Circles that are not used in the formalized statement.  
2280     When you see an error message that contains "unknown identifier", it means that you might have  
2281     forgotten to declare some Points, Lines, or Circles that are used in the formalized statement. Please  
2282     double-check this, or your formalized statement won't compile!

2283     14. Syntax Tip 3: You shouldn't declare any extra Points, Lines, or Circles that are not used in the  
2284     formalized statement. When you see an error message that contains "Unexpected expression", it means that  
2285     you might have declared some extra variables that are not used in the formalized statement. Please  
2286     double-check this, or your formalized statement won't compile!

2287     15. Syntax Tip 4: A formalized angle \angle A:B:C always expects three point identifiers separated by colons  
2288     ! Even though you will see the English expressions like "\angle Y", but "\angle Y" is not a valid  
2289     formalization. You should formalize it into "\angle X:Y:Z" where X, Y, and Z are the three points that  
2290     form the angle. When you see an error message like "unexpected token ':'; expected ':", it means that  
2291     you have not provided the correct number of identifiers for the angle, or you need to surround the angle

2268 with parentheses like (\angle X:Y:Z) to avoid ambiguity. Please double-check this, or your formalized  
 2269 statement won't compile!  
 2270 16. Reasoning: Please think carefully step-by-step and state your intermediate reasoning steps before you semi  
 2271 -formalize each clause when converting the English Statement to Semi-Formalized Structure, formalize  
 2272 each clause when converting the Semi-Formalized Structure into Formalized Structure, and write down the  
 2273 final Formalized Statement. \*\*This is crucial!!! Please do not skip this!!! If there are examples  
 2274 provided by the user, you must strictly follow the format and reasoning in the examples!!!\*\*  
 2275 17. Response Format: Present your formalized Lean expression within triple angle brackets (<<< Lean expression  
 2276 here >>>). Do not add any annotations/explanations, or markdown syntax.  
 2277

## 2277 D.2.2 LEANEUCLIDPLUS ONE-SHOT EXAMPLE

2278 Here is an example:

2280 English Statement:

2281 There are two triangles \triangle GHJ and \triangle IFJ with a shared vertex J. The lines GI and HF  
 2282 intersect at point J. The point J lies between G and I, and also lies between H and F.

2283 Given G J / I J = H J / F J. Complete the proof that \triangle G H J ~ \triangle I F J.

2284 Semi-Formalized Structure:

2285 

```
/*
 To semi-formalize the English Statement into a structure of premises and conclusions, we will examine the
 English Statement sentence by sentence, and break down each sentence into a natural language clause
 that is as atomic as possible.
*/
```

2286 

```
{
 "premises": [
 /*
 The sentence "There are two triangles \triangle GHJ and \triangle IFJ with a shared vertex J." first
 declares 2 triangles \triangle G H J and \triangle I F J.
```

2287 

```
 From the instructions, we know that all points and lines are distinct, so it also declares 5 distinct
 points: H, J, G that form triangle \triangle G H J, and I, F, J that form triangle \triangle I F J.
```

2288 

```
 We can break down the sentence into the following 2 premises:
 */
 "The distinct points G, H, and J form a triangle \triangle G H J",
 "The distinct points I, F, and J form a triangle \triangle I F J",
 /*
```

2289 

```
 For now, the premises above only partially express the fact that points G, H, J, I, F are all distinct
 , and some lines (sides of the triangles) are distinct.
```

2290 

```
 However, we might be able to incorporate and express this distinctness relation in later premises, so
 we don't create additional premises for them right now.
```

2291 

```
 Instead, we will first examine all sentences in the English statement and eventually create premises
 for the remaining unexpressed distinctness relations if necessary.
 /*
```

2292 

```
 /*
 The sentence "The lines GI and HF intersect at point J." first declares 2 lines: GI, HF.
```

2293 

```
 From the instructions, we know that all points and lines are distinct, so it also declares 4 distinct
 points and 2 distinct lines: H, F on line HF, G, I on line GI.
```

2294 

```
 It then states an intersection relation: the lines GI and HF intersect at point J.
```

2295 

```
 If we don't explicitly state the distinctness of the points and lines, it might be interpreted that
 either G and I can be the same point, H and F can be the same point, or GI and HF are the same
 line, which is not what we want.
```

2296 

```
 Therefore, we can break down the sentence into the following 3 premises:
 /*
```

2297 

```
 "H and F are distinct points on line HF",
 "G and I are distinct points on line GI",
 "The distinct lines GI and HF intersect at point J",
 /*
```

2298 

```
 Since we had explicitly expressed that the points G, H, J are mutually distinct, and that the points I
 , F, J are mutually distinct, we have also expressed all distinctness relations between the
 points G, H, J, I, F except for the distinctness relation between G, F, H and I.
```

2299 

```
 If G were the same as F, then GI (FI) would intersect HF at point F as well as point J, which means
 that either J = F or GI = HF, which contradicts the fact that J and F are distinct points or GI
 and HF are distinct lines. Therefore, G and F must be distinct points.
```

2300 

```
 If H were the same as I, following the exact same reasoning, we will get a contradiction. Therefore, H
 and I must be distinct points.
```

2301 

```
 Therefore, we checked that all distinctness relations among the points G, H, I, F are expressed. There
 is only one more distinctness relation left to check: the lines GH and FI are distinct.
```

```

2322
2323 If GH were the same as FI, we would have the intersection of GI and HF, point J also lying on GH (FI),
2324 which contradicts the fact that G, H, and J (I, F, and J) form a triangle. Therefore, GH and FI
2325 must be distinct lines.
2326
2327 Therefore, we checked that all distinctness relations among the points and lines are expressed.
2328 /*
2329 */
2330 The sentence "The point J lies between G and I, and also lies between H and F." states order relations
2331 : the point J is between G and I, and also between H and F.
2332
2333 If we don't explicitly state the distinctness of the points, it might be interpreted that J can be the
2334 same as G or I, or J can be which is not what we want.
2335
2336 Therefore, we have the following 2 premises:
2337 /*
2338 "Among the distinct points G, J, I, the point J is between G and I",
2339 "Among the distinct points H, J, F, the point J is between H and F",
2340 /*
2341 The sentence "Given G J / I J = H J / F J." states a numeric equation between the ratios of line
2342 segments: G J / I J = H J / F J.
2343
2344 Therefore, we have the following premise:
2345 /*
2346 "The ratio of segment GJ to segment IJ equals the ratio of segment HJ to segment FJ, denoted as G J /
2347 I J = H J / F J",
2348],
2349 "conclusions": [
2350 /*
2351 The sentence "Complete the proof that \triangle G H J ~ \triangle I F J." states the goal of the
2352 proof: to show that the triangles \triangle G H J and \triangle I F J are similar.
2353
2354 Therefore, we have the following conclusion:
2355 /*
2356 "The triangles \triangle G H J and \triangle I F J are similar, denoted as \triangle G H J ~ \
2357 \triangle I F J"
2358]
2359 }

2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375

Formalized Structure:
*/
To convert the Semi-Formalized Structure into a Formalized Structure, we will examine the natural language
clauses in the Semi-Formalized Structure one by one, and formalize each into a formal clause that
is well-formed and well-defined according to the guidelines.
{
 "premises": [
 /*
 The clause "The distinct points G, H, and J form a triangle \triangle G H J" has a potential direct
 formalization in the guidelines: `formTriangle`.
 */
 We double check the description of the `formTriangle` relation: "mutually distinct points a, b and c
 form a triangle, where point a and b are on line AB, point b and c are on line BC, point a and c
 are on line CA. The lines AB, BC, and CA must be distinct".
 The mutual distinctness of points in our target clause matches the description of "mutually distinct
 points a, b and c". By definition, the 3 points that form a triangle must be non-collinear,
 which matches the description "The lines AB, BC, and CA must be distinct". Therefore, this
 relation is indeed equivalent to our target clause.

 `formTriangle` takes (a b c : Point) (AB BC CA : Line) as arguments in order, so we can formalize the
 clause as: `formTriangle G H J GH HJ JG`.

 The following clause "The distinct points I, F, and J form a triangle \triangle I F J" can be
 formalized in the exact same way.

 Therefore, we can formalize the first 2 clauses as follows:
 /*
 "formTriangle G H J GH HJ JG",
 "formTriangle I F J IF FJ JI",
 /*
 The clause "H and F are distinct points on line HF" has a potential direct formalization in the
 guidelines: `distinctPointsOnLine`.

 We double check the description of the `distinctPointsOnLine` relation: "points a and b are distinct
 and on line L". It is indeed equivalent to our target clause.

 It takes (a b : Point) (L : Line) as arguments in order, so we can formalize the clause as: `distinctPointsOnLine H F HF`.

 The following clause "G and I are distinct points on line GI" can be formalized in the exact same way.

 Therefore, we can formalize the next 2 clauses as follows:
 /*
 "distinctPointsOnLine H F HF",
 "distinctPointsOnLine G I GI",
]
}

```

```

2376 /*
2377 The clause "The distinct lines GI and HF intersect at point J" has a potential direct formalization in
2378 the guidelines: `twoDistinctLinesIntersectAtPoint'.
2379 We double check the description of the `twoDistinctLinesIntersectAtPoint` relation: "two distinct
2380 lines L and M intersect at point i, i.e. L != M, i lies on both L and M, and `L.intersectsLine M
2381 `". It is indeed equivalent to our target clause since it guarantees that the lines are distinct
2382 .
2383 It takes (L M : Line) (i : Point) as arguments in order, so we can formalize the clause as follows:
2384 */
2385 "twoDistinctLinesIntersectAtPoint GI HF J",
2386 /*
2387 The clause "Among the distinct points G, J, I, the point J is between G and I" has a potential direct
2388 formalization in the guidelines: `between'.
2389 We double check the description of the `between` relation: "mutually distinct points a, b and c are
2390 collinear and ordered. Point b is between point a and c". It is indeed equivalent to our target
2391 clause since it guarantees that the points are mutually distinct.
2392 It takes (a b c : Point) as arguments in order, with Point b being the one that is between Point a and
2393 Point c, so we can formalize the clause as: `between G J I'.
2394 The following clause "Among the distinct points H, J, F, the point J is between H and F" can be
2395 formalized in the exact same way.
2396 Therefore, we can formalize the next 2 clauses as follows:
2397 */
2398 "between G J I",
2399 "between H J F",
2400 /*
2401 The clause "The ratio of segment GJ to segment IJ equals the ratio of segment HJ to segment FJ,
2402 denoted as G J / I J = H J / F J" has a potential direct formalization in the guidelines: `equalLengthRatios'.
2403 We double check the description of the `equalLengthRatios` relation: "equality of two segment-length
2404 ratios: $(|AB| / |CD|) = (|EF| / |GH|)$. This is a real-number equation; no non-zero-length guard
2405 is enforced". It is indeed equivalent to our target clause.
2406 It takes (a b c d e f g h : Point) as arguments in order. Therefore, the parameters a, b corresponds
2407 our concrete arguments G and J, the parameters c, d corresponds to I and J, the parameters e, f
2408 corresponds to H and J, and the parameters g, h corresponds to F and J, so we can formalize the
2409 clause as follows:
2410 */
2411 "equalLengthRatios G J I J H J F J"
2412],
2413 "conclusions": [
2414 /*
2415 The clause "The triangles \triangle G H J and \triangle I F J are similar, denoted as \triangle
2416 G H J ~ \triangle I F J" has a potential direct formalization in the guidelines: `trianglesSimilar'.
2417 We double check the description of the `similar` relation: "triangles ABC and DEF are similar iff: (1)
2418 corresponding angles are equal: \angle BAC = \angle EDF, \angle ABC = \angle DEF, \angle
2419 ACB = \angle DFE; (2) corresponding sides are proportional: |AB|/|DE| = |BC|/|EF| and
2420 |BC|/|EF| = |CA|/|FD|. The correspondence is A<->D, B<->E, C<->F. No non-degeneracy is assumed".
2421 It is indeed equivalent to our target clause.
2422 We check that the point G corresponds to the point I, the point H corresponds to the point F, and the
2423 point J corresponds to the point J.
2424 The `trianglesSimilar` relation takes (a b c d e f : Point) as arguments in order. Therefore, the
2425 parameters a, b, c corresponds our concrete arguments G, H, J, the parameters d, e, f
2426 corresponds to I, F, J, so we can formalize the clause as follows:
2427 */
2428 "trianglesSimilar G H J I F J"
2429 }
2430
2431 Formalized Statement:
2432
2433 Before converting the Formalized Structure into a Formalized statement, we first need to declare all the
2434 geometric objects that are mentioned in the clauses of the Formalized Structure as variables.
2435
2436 According to the guidelines, since all our declared variables will be bounded by a universal quantifier \forall
2437 , we shouldn't declare any extra variables that are not mentioned in any clauses of Formalized
2438 Structure.
2439
2440 We will examine the clauses in the Formalized Structure one by one, and declare variables for the mentioned
2441 geometric objects in order.
2442
2443 The first 2 clauses, `formTriangle G H J GH HJ JG`, and `formTriangle I F J IF FJ JI`, require the declaration
2444 of Points: G, H, J, I, F, and Lines: GH, HJ, JG, IF, FJ, JI.
2445
2446 The next 2 clauses, `distinctPointsOnLine H F HF`, and `distinctPointsOnLine G I GI`, requires the additional
2447 declaration of Lines: HF, GI.
2448
2449 The remaining clauses require no additional declarations.

```

2430 Therefore, we need to declare all and only the following geometric objects as variables:  
 2431  
 2432  $(G H J I F : \text{Point}) (GH HJ JG IF FJ JI HF GI : \text{Line})$   
 2433 Now we can combine the declarations, premises, and conclusions to convert the Formalized Structure to a  
     Formalized Statement. According to the guidelines, we will do the following:  
 2434  
 1. Quantify the declared variables correctly.  
 2. Use the conjunction of all formal clauses in the premises as the antecedent.  
 2436 3. Use the conjunction of all formal clauses in the conclusions as the consequent.  
 4. Connect the antecedent and consequent with an implication ( $\rightarrow$ ).  
 2437 5. Wrap the entire Formalized Statement with triple angle brackets ( $<<< \text{Lean expression here} >>>$ ) for parsing.  
 2438 Finally, we have the Formalized Statement:  
 2439  
 2440  $<<< \forall (G H J I F : \text{Point}) (GH HJ JG IF FJ JI HF GI : \text{Line}), \text{formTriangle } G H J GH HJ JG \backslash\backslash \text{and}$   
      $\text{formTriangle } I F J IF FJ JI \backslash\backslash \text{and } \text{distinctPointsOnLine } H F HF \backslash\backslash \text{and } \text{distinctPointsOnLine } G I GI \backslash\backslash$   
      $\text{land } \text{twoDistinctLinesIntersectAtPoint } GI HF J \backslash\backslash \text{land } \text{between } G J I \backslash\backslash \text{land } \text{between } H J F \backslash\backslash \text{land}$   
      $\text{equalLengthRatios } G J I J H J F J \rightarrow \text{trianglesSimilar } G H J I F J >>>$   
 2442  
 2443 **D.2.3 PROOFNET-HARD INSTRUCTIONS WITH LEARNED ABSTRACTION**  
 2444  
 2445 You are given an English Statement of a mathematical theorem. Target environment: Lean 4.7.0-rc2 with Mathlib4  
     . Do NOT use Lean 3 or deprecated identifiers. Use only current Lean 4/Mathlib4 names and notations.  
 2446  
 2447 Your task is to formalize the English Statement into a formal theorem in Lean 4 using the Mathlib library and  
     no other libraries, strictly adhering to the following formal definitions and guidelines.  
 2448  
 2449 Here are an extra set of helpers you can **\*\*DIRECTLY USE\*\*** in addition to Mathlib:  
 2450  
 2451 `import Mathlib`  
 2452  
 2453 `/-!`  
 2454 This file extends Mathlib with a small, self-contained collection of reusable  
     definitions used in undergraduate-level formalization tasks. The design  
     emphasizes generality, concise naming, and consistency with Mathlib conventions.  
 2455  
 2456 `Contents:`  
 2457  
 2458 1. Iterated square-root map on  $\mathbb{R} \geq 0$  and a recursive sequence:  
     `- `nnrealSqrt2PlusSqrt` :  $x \mapsto \sqrt{2 + \sqrt{x}}$  on  $\mathbb{R} \geq 0$`   
     `- `nnrealSqrt2` : the nonnegative real  $\sqrt{2}$`   
     `- `nnrealIterSqrtSeq` : the sequence defined by  $s_0 = \sqrt{2}$ ,  $s_{n+1} = \sqrt{2 + \sqrt{s_n}}$`   
 2459  
 2460 2. Simple sequence predicates and examples:  
     `- `StrictlyBoundedAboveBy` : strict upper bound predicate for sequences`  
     `- `sqrtSuccDiff` : the real sequence  $n \mapsto \sqrt{n+1} - \sqrt{n}$`   
 2461  
 2462 3. Comparability of topologies:  
     `- `TopologiesComparable` :  $t \leq u \vee t = u \vee u \leq t$`   
 2463  
 2464 4. Subbasis and generated topologies on  $\mathbb{R}$ :  
     `- `IsSubbasisFor` : a family is a subbasis for a topology if the topology is generated from it`  
     `- `LowerLimitSubbasis` and `sorgenfreyTopology` (lower limit / Sorgenfrey line)`  
     `- `KNatRecip`, `KTopologySubbasis`, and `kTopology` (K-topology on  $\mathbb{R}$ )`  
     `- `RationalIooSubbasis` and `realTopologyFromRationalIoo` (topology from rational intervals)`  
 2465  
 2466 5. Pointwise convergence on a set and limit points:  
     `- `UnitInterval` : the type  $\text{Icc } 0 1$`   
     `- `powSeqOnUnitInterval` : the sequence of functions  $x \mapsto x^n$  on the unit interval`  
     `- `powLimitOnUnitInterval` : its pointwise limit`  
     `- `PointwiseConvergesOn` : pointwise convergence on a set`  
     `- `IsLimitPoint` and `LimitPointCompact` : limit points and limit point compactness`  
 2467  
 2468 6. Nested closed nonempty families:  
     `- `NestedClosedNonempty` : antitone closed nonempty families`  
     `- `NestedClosedNonemptyInterNonempty` : their intersection is nonempty`  
 2469  
 2470 All definitions are marked with `@[simp]` as requested.  
 2471  
 2472 `-/`  
 2473  
 2474 `noncomputable section`  
 2475  
 2476 `open Real Filter Set TopologicalSpace`  
 2477 `open scoped NNReal`  
 2478  
 2479 `/-- The function on  $\mathbb{R} \geq 0$  given by  $x \mapsto \sqrt{2 + \sqrt{x}}$ . -/
 2480 @[simp]
 def nnrealSqrt2PlusSqrt (x :  $\mathbb{R} \geq 0$ ) :  $\mathbb{R} \geq 0 :=$ 
 \langle \text{Real.sqrt } ((2 : \mathbb{R}) + \text{Real.sqrt } x), \text{Real.sqrt_nonneg } \rangle  
 2481  
 2482 /-- The nonnegative real number  $\sqrt{2}$ . -/
 @[simp]
 def nnrealSqrt2 :  $\mathbb{R} \geq 0 :=$ 
 \langle \text{Real.sqrt } (2 : \mathbb{R}), \text{Real.sqrt_nonneg } \rangle  
 2483  
 2484 /-- The recursively defined sequence on  $\mathbb{R} \geq 0$ :  
 2485`

```

2484 `s 0 = sqrt(2)` and `s (n+1) = sqrt(2 + sqrt(s n))`. -/
2485 @[simp]
2486 def nnrealIterSqrtSeq : \mathbb{N} -> \mathbb{R} :=
2487 | 0 => nnrealSqrt2
2487 | n+1 => nnrealSqrt2PlusSqrt (nnrealIterSqrtSeq n)

2488 /-- A predicate expressing that a sequence `s : \mathbb{N} -> \alpha` is strictly bounded above by `c`,
2489 i.e. `\\forall n, s n < c`. -/
2490 @[simp]
2490 def StrictlyBoundedAboveBy (\alpha : Type*) [LT \alpha] (s : \mathbb{N} -> \alpha) (c : \alpha) :
2490 Prop :=
2491 \\forall n, s n < c

2492 /-- The real sequence `n |-> sqrt(n + 1) - sqrt(n)`. -/
2493 @[simp]
2493 def sqrtSuccDiff (n : \mathbb{N}) : \mathbb{R} :=
2494 Real.sqrt ((n : \mathbb{R}) + 1) - Real.sqrt (n : \mathbb{R})

2495 /-- Two topologies on the same type are comparable if one is included in the other
2496 (i.e. `t <= u \lor u <= t`). -/
2496 @[simp]
2497 def TopologiesComparable (\alpha : Type*) (t u : TopologicalSpace \alpha) : Prop :=
2497 t <= u \lor u <= t

2498 /-- A family `S : Set (Set X)` is a subbasis for a topology `t` on `X` if
2499 `t` is generated from `S`. -/
2500 @[simp]
2500 def IsSubbasisFor {X : Type*} (S : Set (Set X)) (t : TopologicalSpace X) : Prop :=
2500 t = TopologicalSpace.generateFrom S

2502 /-- The generating family for the lower limit (Sorgenfrey) topology on `\\mathbb{R}`:
2503 all half-open intervals `[a, b)` with `a < b`. -/
2503 @[simp]
2504 def LowerLimitSubbasis : Set (Set \mathbb{R}) :=
2504 {U | \\exists a b : \mathbb{R}, a < b \land U = Set.Ico a b}

2506 /-- The lower limit (Sorgenfrey) topology on `\\mathbb{R}`,
2507 generated by `{{[a, b) | a < b}}`. -/
2507 @[simp]
2508 def sorgenfreyTopology : TopologicalSpace \mathbb{R} :=
2508 TopologicalSpace.generateFrom LowerLimitSubbasis

2509 /-- The set `K := {1/n | n \\in \mathbb{N}^+}` is used in the definition of the K-
2510 topology. -/
2510 @[simp]
2511 def KNatRecip : Set \mathbb{R} :=
2511 {x | \\exists n : \mathbb{N}, x = (1 : \mathbb{R}) / ((n : \mathbb{N}) : \mathbb{R})}

2513 /-- The generating family for the K-topology on `\\mathbb{R}`:
2514 all open intervals `(a, b)` and the punctured intervals `(a, b) \setminus K`. -/
2514 @[simp]
2515 def KTopologySubbasis : Set (Set \mathbb{R}) :=
2515 {U | \\exists a b : \mathbb{R}, a < b \land (U = Set.Ioo a b \lor U = Set.Ioo a b \setminus KNatRecip)}

2516 /-- The K-topology on `\\mathbb{R}` is generated by the subbasis consisting of all sets
2517 of the form `(a, b)` and `(a, b) \setminus K`, where `K = {1/n | n \\in \mathbb{N}^+}`. -/
2517 @[simp]
2518 def kTopology : TopologicalSpace \mathbb{R} :=
2518 TopologicalSpace.generateFrom KTopologySubbasis

2520 /-- The generating family for the topology on `\\mathbb{R}` with a basis of intervals having
2521 rational endpoints: `Ioo a b` with `a, b \\in \mathbb{Q}` and `a < b`. -/
2521 @[simp]
2522 def RationalIooSubbasis : Set (Set \mathbb{R}) :=
2522 {U | \\exists a b : \mathbb{Q}, a < b \land U = Set.Ioo (a : \mathbb{R}) (b : \mathbb{R})}

2523 /-- The topology on `\\mathbb{R}` generated by the subbasis of open intervals with rational endpoints. -/
2523 @[simp]
2524 def realTopologyFromRationalIoo : TopologicalSpace \mathbb{R} :=
2524 TopologicalSpace.generateFrom RationalIooSubbasis

2526 /-- The unit interval `'[0,1]` seen as a subtype of `\\mathbb{R}`. -/
2526 @[simp] abbrev UnitInterval : Type := Set.Icc (0 : \mathbb{R}) 1

2528 /-- The sequence of functions on the unit interval `'[0,1]` given by
2528 `(powSeqOnUnitInterval n) x = x^n`. -/
2528 @[simp]
2529 def powSeqOnUnitInterval : \mathbb{N} -> UnitInterval -> \mathbb{R} :=
2529 fun n x => (x : \mathbb{R}) ^ n

2531 /-- The pointwise limit function on `'[0,1]` associated with the sequence `x |-> x^n`:
2531 it is `0` for `x < 1` and `1` otherwise (in particular at `x = 1`). -/
2531 @[simp]
2532 def powLimitOnUnitInterval : UnitInterval -> \mathbb{R} :=
2532 fun x => if (x : \mathbb{R}) < 1 then 0 else 1

2535 /-- Pointwise convergence on a set: a sequence of functions `F : \mathbb{N} -> X -> \beta` converges
2535 pointwise on `S` to `g : X -> \beta` if `\\forall x \\in S, \\lim_{n \\rightarrow \\infty} (F n x) = g x`).
2535 -/
2535 @[simp]
2536 def PointwiseConvergesOn {X \beta : Type*} [TopologicalSpace \beta]
2536 (F : \mathbb{N} -> X -> \beta) (g : X -> \beta) (S : Set X) : Prop :=

```

```

2538 \forall x \in S, Tendsto (fun n => F n x) atTop (nhds (g x))
2539
2540 -- A point `x` is a limit point (accumulation point) of a subset `A` of a topological space `X`
2541 if every neighborhood of `x` meets `A \ {x}`. -/
2542 @[simp]
2543 def IsLimitPoint {X : Type*} [TopologicalSpace X] (A : Set X) (x : X) : Prop :=
2544 \forall U \in nhds x, (U \cap (A \ {x})).Nonempty
2545
2546 -- A topological space is limit point compact if every infinite subset has a limit point. -/
2547 @[simp]
2548 def LimitPointCompact {X : Type*} [TopologicalSpace X] : Prop :=
2549 \forall A : Set X, A.Infinite -> \exists x : X, IsLimitPoint A x
2550
2551 -- A nested (decreasing) sequence `C : \mathbb{N} \rightarrow Set X` of closed, nonempty sets in a
2552 -- topological space:
2553 Antitone C \land (\forall n, IsClosed (C n)) \land (\forall n, (C n).Nonempty). -/
2554 @[simp]
2555 def NestedClosedNonempty {X : Type*} [TopologicalSpace X] (C : \mathbb{N} \rightarrow Set X) : Prop :=
2556 Antitone C \land (\forall n : \mathbb{N}, IsClosed (C n)) \land (\forall n : \mathbb{N}, (C n).Nonempty)
2557
2558 -- The property that every nested (decreasing) sequence of closed nonempty subsets of `X`
2559 -- has nonempty intersection `\\bigcap n, C n`. -/
2560 @[simp]
2561 def NestedClosedNonemptyInterNonempty {X : Type*} [TopologicalSpace X] : Prop :=
2562 \forall (C : \mathbb{N} \rightarrow Set X),
2563 Antitone C ->
2564 (\forall n : \mathbb{N}, IsClosed (C n)) ->
2565 (\forall n : \mathbb{N}, (C n).Nonempty) ->
2566 (\bigcap n, C n).Nonempty
2567
2568 1. Target & Environment
2569 - Formalize an English statement into Lean 4 using **ONLY** Mathlib
2570 - **NO** other libraries or import should be used.
2571 - Environment: Lean 4.7.0-rc2 with Mathlib4. Use only current Lean 4/Mathlib4 identifiers. Do NOT use Lean
2572 3-era names.
2573
2574 2. Forbidden Lean 3-Era Identifiers & Notations
2575 - Identifiers: `Convergent`, `Function.IsFieldHom`, `QuotientGroup.quotient`, `Metric.bounded`, `IsPerfect
2576 `, `Real.cbtr`.
2577 - Notations: `\\mathbb{R}^m`, `Z[i]`, raw `int x in \\mathbb{R}`, `\\ldots`, binder shorthand `\\forall x y
2578 \\in S, \\ldots`, negative exponents with `^` (e.g., `^(-p)`).
2579 - Use `RingHom`/*`AlgHom` (not `Function.IsFieldHom`).
2580 - Use `Subgroup.Quotient` (not `QuotientGroup.quotient`).
2581 - Use `Bornology.IsBounded E` (not `Metric.bounded E`).
2582 - Avoid `Real.cbtr`; use `Real.rpow` or rational exponents.
2583
2584 3. Header Format:
2585 - You need to generate **BOTH** the header and the theorem in your final response
2586 - The **ONLY** import statement you should generate is `import Mathlib`.
2587 - After that, write `open ...` and `open scoped ...` for the namespaces you need to use in the theorem.
2588 - **MAKE SURE** that the names and notations you are using are opened in the header!!! Please check this if
2589 you receive syntax errors!!!
2590
2591 4. Theorem Format: ALWAYS include theorem name, and it MUST be exactly `thm_Q`.
2592 - Required format: `theorem thm_Q (params) : conclusion := by sorry`
2593 - You **MUST** use `by sorry` as the placeholder. Do **NOT** generate a proof!!!
2594 - Note that some formalizations of a statement might not have parameters/binders, only the type/goal. For
2595 example, "Prove that $\\sqrt[3]{2}+\\sqrt[3]{3}$ is irrational." can be formalized as `theorem thm_Q :
2596 Irrational (2^(1:\\mathbb{R} / 3)) + 3^(1:\\mathbb{R} / 3)) := by sorry`
2597 - You should try to use the binder/parameter form as much as possible. For example, try to convert
2598 existential statements to universal statements so that you can use the binder/parameter form.
2599
2600 5. Parameter Format: Use explicit parentheses for ALL parameters/binders `(\Omega : Set \mathbb{C}) (f : \mathbb{C} -> \mathbb{C}) (h : IsOpen \Omega)`. NEVER use implicit `(\Omega)`.
2601
2602 6. Parameter/Binder Naming:
2603 - For object names, use the **EXACT SAME** names as the in problem text. For example, if the problem
2604 mentions "complex function f" then you should name the corresponding parameter `f` like `(f : \mathbb{C} -> \mathbb{C})`.
2605 - For hypothesis names, use the object name with a prefix `h` and index them from 1. For example, if the
2606 problem mentions "complex function f that is holomorphic on X" then you should name the corresponding
2607 hypothesis `hf_1` like `(hf_1 : DifferentiableOn \mathbb{C} f X)`.
2608
2609 7. Reasoning
2610 - Please think carefully step-by-step and state your intermediate reasoning steps before write down the
2611 final Formalized Statement.
2612 - This is **EXTREMELY CRUCIAL!!!** Please do not skip this!!! **PAY EXTRA ATTENTION TO THIS!!!**
2613 - If there are examples provided by the user, you must strictly follow the format and reasoning in the
2614 examples!!! **THINK EXTRA HARD!!!**
2615
2616 8. Response Format: Return the header and the theorem together, surrounded by triple angle brackets <<< import
2617 Mathlib
2618 open ...
2619 open scoped ...
2620
2621 theorem thm_Q ... := by sorry >>>.
2622
2623 Do **NOT** include **ANY** comments or proof tactics in your final response!!! **PAY EXTRA ATTENTION TO THIS
2624 !!!**

```

```

2592
2593 Example of CORRECT output (declare all variables, name must be thm_Q):
2594 <<< import Mathlib
2595 open ...
2596 open scoped ...
2597
2598 theorem thm_Q : \forall N : \mathbb{N}, \exists n \geq N, (3*n+1).Prime \land (3*n+1) \geq N := by sorry
2599 >>>
2600
2601 Example of INCORRECT output (wrong theorem name, missing variable N declaration, includes comments, missing `by sorry`):
2602 <<< import Mathlib
2603 open ...
2604 open scoped ...
2605
2606 -- some random comment
2607 theorem some_random_name : \exists n \geq N, (3*n+1).Prime \land (3*n+1) \geq N := >>>
2608
2609
2610 D.2.4 PROOFNET-HARD ONE-SHOT EXAMPLE
2611
2612 Here is an example:
2613 English Statement:
2614 Let $\mathbb{U} \subset \mathbb{C}$ be a (non-empty) connected open set and let $\{f_n\}$ be a sequence of holomorphic
2615 functions defined on \mathbb{U} . Suppose that $\{f_n\}$ converges uniformly to a function f on every compact
2616 subset of \mathbb{U} . Show that f is holomorphic in \mathbb{U} .
2617
2618 Semi-Formalized Structure:
2619
2620 /*
2621 To semi-formalize the English Statement into a nested structure of quantifications, premises and
2622 conclusions, we will examine the English Statement sentence by sentence, and break down each
2623 sentence into natural language clauses that are as atomic as possible.
2624 */
2625 {
2626 /*
2627 The sentence "Let $\mathbb{U} \subset \mathbb{C}$ be a (non-empty) connected open set and let $\{f_n\}$ be a sequence of holomorphic
2628 functions defined on \mathbb{U} ." indicates that the choice of \mathbb{U} and $\{f_n\}$ are arbitrary,
2629 so we should choose the universal quantifier for the entire statement that quantifies over \mathbb{U} ,
2630 $\{f_n\}$, and possibly other variables.
2631
2632 Since $\{f_n\}$ is a sequence of functions with certain properties, we will further break it into a nested sub-
2633 -structure. Now, we will first universally quantify over \mathbb{U} as follows:
2634 */
2635 "quantification": "for all \mathbb{U} , $\{f_n\}$, possibly other variables",
2636 "premises": [
2637 /*
2638 The sentence first declares that \mathbb{U} is a subset of the complex plane \mathbb{C} , that it is a non-
2639 empty, a connected, and an open, so we have the following 4 premises:
2640 */
2641 " \mathbb{U} is a subset of the complex plane \mathbb{C} ",
2642 " \mathbb{U} is non-empty",
2643 " \mathbb{U} is connected",
2644 " \mathbb{U} is open",
2645 /*
2646 The sentence also declares that $\{f_n\}$ is a sequence of complex functions, so we have the following
2647 premise:
2648 */
2649 " $\{f_n\}$ is a sequence of functions from \mathbb{C} to \mathbb{C} ",
2650 /*
2651 Now we can break down the specification on $\{f_n\}$, a sequence of holomorphic functions defined on \mathbb{U} .
2652 Recall our mathematical knowledge, a sequence is countably indexed, which means that for all $i \in \mathbb{N}$,
2653 f_i is holomorphic on \mathbb{U} .
2654
2655 By the definition of holomorphic function, each f_i is a function from \mathbb{U} to \mathbb{C} and is
2656 complex-differentiable at every point of \mathbb{U} .
2657
2658 Therefore, we break it into a nested sub-structure by universally quantifying over i as follows:
2659 */
2660 {
2661 "quantification": "for all i ",
2662 "premises": [
2663 " i is a natural number i.e. $i \in \mathbb{N}$ ",
2664],
2665 "conclusions": [
2666 " f_i in the sequence $\{f_n\}$ is holomorphic on \mathbb{U} , which means that f_i is complex-
2667 differentiable at every point of \mathbb{U} "
2668]
2669 },
2670 /*
2671 The next sentence "Suppose that $\{f_n\}$ converges uniformly to a function f on every compact subset of
2672 \mathbb{U} " indicates that there exists a function f with a certain property, which indicates a
2673 nested existential quantifier over f .
2674
2675 However, our conclusion "Show that f is holomorphic in \mathbb{U} " requires f to be declared beforehand.
2676 Therefore, the correct conclusion should really be "Show that if there exists a function f in

```



```

2700
2701 /*
2702 The next premise is "U is open". Check the provided helpers and recall our knowledge of the required
2703 versions of Lean and Mathlib in the guidelines, there is a direct corresponding formal relation
2704 `IsOpen`. Therefore, we can formalize this property of U as follows:
2705 */
2706 "(hU_3 : IsOpen U)",
2707 /*
2708 The next premise is "f_n is a sequence of complex functions". Check the provided helpers and recall
2709 our knowledge of the required versions of Lean and Mathlib in the guidelines, since a sequence
2710 is countably indexed, we can say that f_n is of the curried function type `\\mathbb{N} -> \\mathbb{C}`.
2711 When we want to talk about the i-th function in the sequence, we
2712 can use the function application notation `f_n i`.
2713
2714 Therefore, we can formalize this premise as follows:
2715 */
2716 "(f_n : \\mathbb{N} -> \\mathbb{C} -> \\mathbb{C})",
2717 /*
2718 The next premise is a universally quantified statement (a sub-structure in our semi-formalized
2719 structure) about each f_i in the sequence f_n: "for all $i \in \mathbb{N}$ we have "f_i is
2720 holomorphic on U". This means that f_i is complex-differentiable at
2721 every point of U.
2722
2723 For "i being a natural number", check the provided helpers and recall our knowledge of the required
2724 versions of Lean and Mathlib in the guidelines, we can say that i is of type `\\mathbb{N}`, so
2725 we can formalize the quantification over i as `\\forall i : \\mathbb{N}`.
2726
2727 For "f_i is complex-differentiable at every point of U", check the provided helpers and recall our
2728 knowledge of the required versions of Lean and Mathlib in the guidelines, there is a direct
2729 corresponding formal relation `DifferentiableOn` to talk about complex-differentiability by
2730 passing `\\mathbb{C}` as the first argument.
2731
2732 Therefore, we can formalize this property of f_i as follows:
2733 */
2734 "(hf_n_1 : \\forall i : \\mathbb{N}, DifferentiableOn \\mathbb{C} (f_n i) U)",
2735 /*
2736 The next premise is "f is a function from $\\mathbb{C}$ to $\\mathbb{C}$". Check the provided helpers
2737 and recall our knowledge of the required versions of Lean and Mathlib in the guidelines, we can
2738 say that f is of the function type `\\mathbb{C} -> \\mathbb{C}`.
2739
2740 Therefore, we can formalize this premise as follows:
2741 */
2742 "(f : \\mathbb{C} -> \\mathbb{C})",
2743 /*
2744 The next premise is a universally quantified statement (a sub-structure in our semi-formalized
2745 structure) about each subset X of U: "for all subsets X of U, if X is compact, then
2746 the sequence f_n converges uniformly to f on X".
2747
2748 For "X being a subset of U", check the provided helpers and recall our knowledge of the required
2749 versions of Lean and Mathlib in the guidelines, there is a direct corresponding formal notation
2750 `\\subseq`. However, according to the guidelines, we should always declare the type of any
2751 variable first. Therefore, we can formalize the quantification over X as `\\forall X : Set \\mathbb{C}` and the
2752 subset relation as `X \\subseq U`.
2753
2754 For "X being compact", check the provided helpers and recall our knowledge of the required
2755 versions of Lean and Mathlib in the guidelines, there is a direct corresponding formal relation
2756 `IsCompact`. Therefore, we can formalize this property of X as follows as `IsCompact X`.
2757
2758 For "the sequence f_n converges uniformly to f on X", check the provided helpers and recall our
2759 knowledge of the required versions of Lean and Mathlib in the guidelines, there is a direct
2760 corresponding relation `TendstoUniformlyOn F f 1 X` for `F : \\iota -> \\alpha -> \\beta` and
2761 `f : \\alpha -> \\beta` along a filter `1`. Here `F` is our sequence of functions `\\lambda
2762 n x, f_n n x`, the limit is `f`, the index filter is `atTop` (we can directly use this name
2763 since we opened the namespace `Filter`), and the set is `X`. We can formalize the conclusion of
2764 this quantified statement as `TendstoUniformlyOn (\\lambda n x => f_n n x) f atTop X`.
2765
2766 Therefore, assembling every piece together, we can formalize this universally quantified statement as
2767 follows:
2768 */
2769 "(hf_1 : \\forall X : Set \\mathbb{C}, X \\subseq U \\land IsCompact X -> TendstoUniformlyOn (\\
2770 lambda n x => f_n n x) f atTop X)",
2771],
2772 "conclusions": [
2773 /*
2774 The conclusion is "f is holomorphic on U". This means that f is complex-differentiable at every
2775 point of U. Check the provided helpers and recall our knowledge of the required versions of
2776 Lean and Mathlib in the guidelines, there is a direct corresponding formal relation
2777 `DifferentiableOn` to talk about complex-differentiability by passing `\\mathbb{C}` as the first
2778 argument.
2779
2780 Therefore, we can formalize this property of f as follows:
2781 */
2782 "DifferentiableOn \\mathbb{C} f U",
2783]
2784
2785 Formalized Statement:
2786
2787 Now we can convert the Formalized Structure to a Formalized Statement. According to the guidelines, we will do
2788 the following:
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
3999

```

2754 1. Assemble the quantification, premises, and conclusions properly into a Formalized Statement. Name the  
 2755 theorem following the guidelines as `thm\_Q'. Use the correct syntax if we are using the binder form.  
 2756 2. Wrap the entire Formalized Statement with triple angle brackets (<<< Lean expression here >>>) for parsing.  
 2757 Finally, we have the Formalized Statement:  
 2758 <<< theorem thm\_Q (U : Set  $\mathbb{C}$ ) (hU\_1 : Nonempty U) (hU\_2 : IsConnected U) (hU\_3 : IsOpen U) (f\_n :  $\mathbb{C} \rightarrow \mathbb{C}$ ) (hf\_n\_1 :  $\forall n : \mathbb{N}$ , DifferentiableOn  $\mathbb{C}$  (f\_n n) U) (f :  $\mathbb{C} \rightarrow \mathbb{C}$ ) (hf\_1 :  $\forall x : \mathbb{C}$ , X  $\subseteq U$   $\wedge$  IsCompact X  $\rightarrow$  TendstoUniformlyOn ( $\lambda n x \mapsto f_n n x$ ) f atTop X) : DifferentiableOn  $\mathbb{C}$  f U := by sorry >>>  
 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