LEANAIDE

A statement autoformalisation tool for Lean

EuroProofNet Workshop 2023

CONTRIBUTORS

- Siddhartha Gadgil
- Anand Rao Tadipatri
- Ayush Agrawal
- Ashvni Narayanan
- Navin Goyal

OVERVIEW

- `LeanAIde` is a tool to translate mathematical statements from natural language to Lean code.
- The tool is itself written in `Lean 4`.
- At its core, `LeanAIde` relies on a large language model for translation.
- Various optimisations to the input and output of the language model are used to push up the success rate of translation.

PROMPTING

The prompting style used to query a language model can have a strong effect on the output.

A few possible prompting styles for autoformalisation include:

- Direct (zero-shot) prompting
- (Fixed) few-shot prompting
- Input-dependent prompting

THE DESIGN

- Receive the input statement from the user through the Lean editor.
- Gather documentation strings from `mathlib` with similar content.
- Assemble a prompt from these doc-strings and query the language model.
- Post-process the outputs and retain only those corresponding to well-formed Lean expressions.
- Pick an output representing the most common translation and display it in the Lean editor.

SENTENCE EMBEDDINGS

Sentence embeddings are numerical representations of text as vectors of real numbers in a way that captures semantic relationships between them.

The embedding of the input statement is computed (using OpenAI embeddings) and compared with stored embeddings of `Mathlib` doc-strings to identify the most similar ones.

PROMPTING

The prompt to the language model is assembled from the sentence embeddings as an alternating dialogue of doc-strings (“from the user”) and their corresponding Lean formal statements (“from the assistant”).

This is sent as a query to the `OpenAI GPT-3.5 Turbo` or `GPT-4` language model via an API call.

Additional configuration options permit adding a few fixed examples to the prompt and also using theorems with doc-strings from the current editor window.

ELABORATION FILTERING

Additionally, we retain only those outputs of the language model that correspond to well-formed Lean expressions.

As Lean is a dependently typed language, this is a very strong condition.

OUTPUT

After post-processing and filtering, the final output is picked by *majority voting*, i.e.,

- the statements are clustered by proved equivalence using the `aesop` automation tool and
- a representative of the most common translation is then presented to the user.

EVALUATION

The `LeanAIde` tool is tested against two datasets:

- A custom data-set of around 120 theorem statements at the undergraudate level
- The `ProofNet` benchmark for statement autoformalisation

CUSTOM DATASET

The custom data-set of 120 statements is split into three categories:

- normal statements
- “silly” statements
- false statements

The last two categories are specifically to guard against data contamination.

PROOFNET

A benchmark for statement autoformalisation consisting of 371 theorem statements drawn from various undergraduate pure mathematics textbooks.

RESULTS

Parameters: 20 input-dependent prompts, 10 outputs per sentence, temperature 0.8

	Total	Number elaborated	Number correct
Normal statements	40	37	36
Silly statements	40	39	36

	Total	Number elaborated	Number correct
False statements	37	31	28

Overall success rate: 85%

PROOFNET RESULTS

Total	Number elaborated	Number correct
100	69	37

SUMMARY

`LeanAIde` is a tool for translating natural language theorem statements to Lean code, with a success rate high enough to be of possible practical use.

The tool crucially relies on several distinctive features of the Lean theorem prover, including its programming and meta-programming capabilities and its the vast and unified mathematics library.

AI AND PROOF ASSISTANTS

There is potential for combining languages models with proof assistants for tasks such as

- Autoformalisation
- Code completions and debugging
- Navigating libraries of formal mathematics
- Suggesting new lemmas during formalisation

Such tools can make formalisation of mathematics vastly more approachable.

REFERENCES

- Zhangir Azerbayev and Edward W. Ayers. lean-chat: user guide. Lean. 2023. url: https://github.com/zhangir-azerbayev/lean-chat.
- Zhangir Azerbayev et al. ProofNet: Autoformalizing and Formally Proving Undergraduate-Level Mathematics. 2023. arXiv: 2302.12433 [cs.CL].
- Naman Jain et al. "Jigsaw: Large language models meet program synthesis". In: Proceedings of the 44th International Conference on Software Engineering. 2022, pp. 1219–1231.

- Albert Q Jiang et al. “Draft, sketch, and prove: Guiding formal theorem provers with informal proofs”. In: arXiv preprint arXiv:2210.12283 (2022).
- Leonardo de Moura and Sebastian Ullrich. “The lean 4 theorem prover and programming language”. In: Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12– 15, 2021, Proceedings 28. Springer. 2021, pp. 625–635.
- Arvind Neelakantan et al. “Text and code embeddings by contrastive pre- training”. In: arXiv preprint arXiv:2201.10005 (2022).
- OpenAI. GPT-4 Technical Report. 2023. arXiv:

- Qingxiang Wang et al. “Exploration of neural machine translation in autoformalization of mathematics in Mizar”. In: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs. 2020,pp. 85–98.
- Yuhuai Wu et al. “Autoformalization with large language models”. In: Advances in Neural Information Processing Systems 35 (2022), pp. 32353–32368.
- Jannis Limperg and Asta Halkjær From. “Aesop: White-Box Best-First Proof Search for Lean”. In: Proceedings of the 12th ACM SIGPLAN International Conference on Certified Programs and