Capstone Three Project Proposal: Empowering theorem-proving AI to generate conjectural lemmas.

Matt Feller

1 Problem statement

The central goal of this project is to train a generative AI model which conjectures useful lemmas during proof search in the theorem-prover Lean. Such a tool could be helpful for human Lean users, but the main idea is to incorporate conjectural lemmas into the automatic prover ReProver in the LeanDojo framework.

2 Background

Although LLMs such as ChatGPT have recently become popular for generating code, the output tends to require substantial checking and debugging by humans. In general, generating reliable code is still out of reach for current AI models. Interactive theorem provers such as Lean provide an ideal setting for exploring improvements in generating code, in part because of the freely available repository of proofs from the efforts at formalizing math (specifically, mathlib4 in Lean 4), but also for the simple fact that theorem prover code verifies itself.

Lean, as an interactive theorem prover, is both a programming language and a setting for writing proofs. One can define variables, functions, and such just like an ordinary programming language, but its primary use is to write formal theorem statements and provide formal proofs. When writing a proof, a second window tracks the current proof state, keeping track of what is known/assumed and what is left to be proved. With every new line of code added to the proof, the proof state is updated. The user adds lines of code until the goal state is reached without errors, at which point the theorem has been formally proved.

There are a number Lean theorem-proving AI models out there, but none achieves a success rate above 42% on the miniF2F benchmark, with the number being even lower for models that do not use reinforcement learning. One of the best and most recent non-RL models is LeanDojo's open-source ReProver, which achieves 26.5% success on miniF2F.

3 Dataset and methods

The LeanDojo open-source framework comes with substantial training and test datasets, and the miniF2F benchmark is available freely online as well.

Our approach is to start by leverage the trained premise retriever in ReProver into a premise generator. Their premise retriever finds and ranks the known lemmas and theorems (the "premises") which seem most useful in the proof, so we plan to use that as a starting point for a model which outputs new, unproved (and potentially untrue) lemmas that seem the most useful for making progress in the proof. We expect that we will need to fine-tune this model to prefer lemma which seem likely to be true and easy to prove.

The current ReProver algorithm consists of a best-first search, where each step involves predicting the next line of code based on the current proof state and a retrieved list of premises. Our plan is to modify this step by having our model generate conjectures and have them considered alongside the retrieved premises when predicting the next line of code. If a proof succeeds using any conjectures, then it attempts to prove those conjectures (and perhaps continues to search for a proof with fewer or no new conjectures). The exact implementation will depend on some experimentation.

4 Potential outcomes

The most desirable outcome is that our updated ReProver outperforms the current version, proving more theorems with similar computational costs. What may be more likely to occur is that our model is unable to reproduce some of the original models proofs due to the complexity introduced, but that possibly there are other theorems that the original model could not tackle where our model is able to succeed; such a result would suggest that further investigations could lead to a Goldilocks model which can replicate the successes of both models. And finally, a completely negative result where introducing conjectures is only a distraction and performance only goes down is still an interesting result; this is an experiment, after all! Plus, even in that case, perhaps a human user could benefit from the suggested conjectures, making our trained model useful in its own right.