Proof Explainer Project

Advancing Proof Comprehension

Vadim Zaliva | vadim@zaliva.org

The Problem

Formal proofs using Proof Assistants such as Coq, Lean, and Isabelle have become important design artefacts of modern high-assurance software systems. Along with the traditional source code base, proof scripts must be reviewed, documented, and maintained. As different groups of proof engineers work on them over time, proof scripts' documentation and overall readability become crucial.

Unfortunately, proof scripts are generally not easily human-readable. Consider the following example of a textbook [1] proof of Gauss' Formula in Coq, and compare it with the explanation generated by the LLM, shown in Appendix A.

```
Example gauss n :
  \sum_(0 <= i < n.+1) i = n * n.+1 %/ 2.
Proof.
  elim: n =>[|n IHn]; first by apply: big_nat1.
  rewrite big_nat_recr //= IHn addnC -divnMDl //.
  by rewrite mulnS muln1 -addnA -mulSn -mulnS.
Qed.
```

The difficulty of human comprehensibility goes beyond the obscure syntax and is inherent to tactic languages. Each tactic operates on an implicit proof state, which is invisible unless you step through the script in the proof assistant. Even if the proof state is available, it could be large, making it difficult to focus on the relevant part of each step. Some features of proof assistants allow for writing more human-readable proofs, but proof writers often ignore them in favour of brevity and expediency. One example is automatic variable naming. While tactic languages support some structuring of the proof script to reflect the branching of the proof tree, these features are not always rigorously used by human proof writers, and their expressivity is limited.

Proposal

We propose to develop a tool to explain and document Coq proofs with the help of LLMs. When applied to an already proven lemma, the Proof Explainer will produce standalone documentation or annotated and restructured for readability proof script. It could be made accessible via a simple web interface or integrated into the proof assistant's IDE (e.g. as a VS Code plugin). The command line version could be used in a build or continuous integration process to generate online documentation or to annotate proof scripts.

It would be immediately helpful to a wide array of users. Professional proof engineers could use it to examine and understand existing proof code bases. The generated documentation could be used to onboard new team members and conduct code reviews. It would immediately benefit students of formal methods and proof engineering by allowing to break down and understand the intricacies of existing proofs. It could be an invaluable educational tool for learning best practices and building confidence in engaging with real-world proof codebases.

The approach

The key observation is that proof explanation requires insight into the proof state, as maintained by a proof assistant. Our tool will integrate a proof assistant to access this information. We can use proof assistant APIs to disambiguate syntax and access the per-step proof state, definitions, and types. When proof automation is used, we can unroll the automation steps to access their intermediate results.

The second insight is that current LLMs have a limit on a LLM's context window, and feeding the whole proof along with all related definitions is impractical. This calls for carefully curating the information supplied to the model at each reasoning step. We can supply only necessary definitions, strip irrelevant parts of the proof context, or the calculated proof stated diffs between the steps.

Additionally, we can leverage different models and arbitrage or verify their results. Using the well-established "AI agents" approach, we can split reasoning into several steps and process each via separate LLM queries.

Finally, we can perform multi-pass elaboration for projects containing reusable lemmas, first processing the auxiliary lemmas and later using AI-generated summaries of already processed lemmas accessible via retrieval-augmented generation (RAG) or similar techniques.

In the initial project stage, we would focus first on the Rocq proof assistant, which is mature and has good automation means (LSP protocol server, plugin API, and command-line tools). Many Rocq proofs are published as open source, which we can use for training, testing, and validation. Our preliminary experiments show that popular LLM models like OpenAIi o1 and Anthropic Claude 3.5 Sonnet can perform basic Rocq proof elaboration, which we can use as a baseline.

Key Differentiators and Related Work

Several recent projects have used LLMs for formal proofs. For example,TBD..... Most attempt to synthesise proofs from scratch or aid in repairing existing proofs. Given the current early state of LLMs, these are ambitious goals. We believe that to reach the ultimate goal of fully automated proof synthesis, merely promoting or guiding existing models is insufficient, and a deeper integration into the LLM reasoning process is required, as, for example, proposed in [2]. [more citations]. The required approaches would likely be costly, with prices comparable to training new LLMs. Our goals are more modest, and we are trying to play on the strength of existing models in knowledge summarisation and presentation, where they are already showing promising results.

Budget & Estimates

This is a relatively small-scale project. We estimate 6-8 months with a budget of approximately $20K, covering the salary of one full-time research engineer and LLM usage fees. The involvement of the PI (and potentially a co-PI) is provided at no cost. References

A Appendix: Explained Coq Proof of Gauss's Summation Formula

Statement and Proof Script

```
Example gauss n :
  \sum_(0 <= i < n.+1) i = n * n.+1 %/ 2.
Proof.
  elim: n => [|n IHn]; first by apply: big_nat1.
  rewrite big_nat_recr //= IHn addnC -divnMDl //.
  by rewrite mulnS muln1 -addnA -mulSn -mulnS.
Qed.
```

Overall Summary of the Proof

- **Goal:** Prove that the sum of the first `n+1` natural numbers

$$0 + 1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

- **Approach:** Use induction on `n`. The base case `n = 0` follows from a standard library lemma about sums up to `1`. For the inductive step, the sum $\sum_{0 \le i < n+2} i$ is split into $\sum_{0 \le i < n+1} i + (n+1)$. Applying the induction hypothesis and then performing straightforward algebraic rewriting yields the closed-form expression $\frac{n(n+1)}{2}$.

Step-by-Step Explanation

1. **Induction on n:**

   ```
   elim: n => [|n IHn];
   ```

   This asks Coq to prove the statement by structural induction on `n`. It generates two subgoals: the base case (when `n` = 0) and the inductive step (assuming the statement for `n` and proving it for `n+1`).

2. **Base case:**

   ```
   first by apply: big_nat1.
   ```

   Here, `big_nat1` is a lemma from the library that states $\sum_{0 \le i < 1} i = 0$. This immediately solves the base case, since $\frac{0 \cdot (0+1)}{2} = 0$.

3. **Rewriting the sum:**

   ```
   rewrite big_nat_recr //= IHn addnC -divnMDl //.
   ```

   `big_nat_recr` rewrites the sum $\sum_{0 \le i < (n+1).+1} i$ into $\sum_{0 \le i < n+1} i + (n+1)$. The `//=` notation triggers simplifications. We then apply the induction hypothesis `IHn` (which replaces $\sum_{0 \le i < n+1} i$ with $\frac{n(n+1)}{2}$) and use lemmas like `addnC` (commutativity of addition) and `-divnMDl` (rewriting a division expression) to structure the expression.

4. **Finishing the algebra:**

   ```
   by rewrite mulnS muln1 -addnA -mulSn -mulnS.
   ```

   These final rewrite steps handle normal algebraic identities in $\mathbb{N}$, such as distributing multiplication over addition (`mulnS`), rearranging sums (`-addnA` for associativity), and so forth. After these rewrites, Coq recognises the exact target form $\frac{n(n+1)}{2}$.

Thus, by induction, we conclude that

$$\sum_{0 \le i < n+1} i = \frac{n(n+1)}{2},$$

as required.

References

[1] Mahboubi, A., and Tassi, E. Mathematical components. *Online book* (2021).

[2] Park, K., Wang, J., Berg-Kirkpatrick, T., Polikarpova, N., and D'Antoni, L. Grammar-aligned decoding, 2024.