Learning to Format Coq Code Using Language Models

Pengyu Nie[1], Karl Palmskog[2], Junyi Jessy Li[1], and Milos Gligoric[1]

[1] The University of Texas at Austin, Austin, TX, USA
[2] KTH Royal Institute of Technology, Stockholm, Sweden
pynie@utexas.edu, palmskog@acm.org, jessy@austin.utexas.edu, gligoric@utexas.edu

Should the final right bracket in a `Record` declaration be on a separate line? Should arguments to `rewrite` be separated by a single space? Coq code tends to be written in distinct manners by different people and teams. The expressiveness, flexibility, and extensibility of Coq's languages and notations means that Coq projects have a wide variety of recognizable coding styles, sometimes explicitly documented as conventions on naming and formatting. In particular, even inexperienced users can distinguish vernacular using the standard library and plain Ltac from idiomatic vernacular using the Mathematical Components (MathComp) library and SSReflect.

While coding conventions are important for comprehension and maintenance, they are costly to document and enforce. Rule-based formatters, such as Coq's beautifier, have limited flexibility and only capture small fractions of desired conventions in large verification projects. We believe that application of *language models*—a class of Natural Language Processing (NLP) techniques for capturing regularities in corpora—can provide a solution to this conundrum [1]. More specifically, we believe that an approach based on automatically learning conventions from existing Coq code, and then suggesting idiomatic code to users in the proper context, can be superior to manual approaches and static analysis tools—both in terms of effort and results.

As a first step, we here outline initial models to learn and suggest *space formatting* in Coq files, with a preliminary implementation for Coq 8.10, and evaluated using on a corpus based on MathComp 1.9.0 which comprises 164k lines of Coq code from four core projects [3].

Language Models for Coq Formatting

Natural language has repeating patterns which can be *predicted* statistically at the level of, say, individual words with high accuracy. Programming languages have similar predictability, usually called *naturalness*, which can be exploited to perform a variety of software engineering tasks [1]. We consider, from this view, the problem of predicting spacing between tokens obtained from Coq's lexer. For example, according to MathComp's contribution guide, there should be no space between the tactic tokens `move` and `=>`, which we can learn by observing the relative locations of the two tokens in a large Coq corpus adhering to the conventions.

n-gram model: We constructed a baseline model based on predicting the next token after observing the $n-1$ previous tokens, as often used in NLP and software engineering. To capture formatting, we inserted special tokens holding spacing information before each token.

Neural model: We constructed a sophisticated model based on bi-directional recurrent neural networks [4]. The model embeds Coq tokens and spacing information into vectors, and predicts token formatting using the embedding vectors of both the left-hand and right-hand context.

Preliminary Implementation and Evaluation

To implement learning and suggesting of spacing in Coq files based on our models, we modified the SerAPI library [2] to serialize tokens in Coq files, organized at the sentence level. We then serialized all sentences in our MathComp corpus, and extracted information on token kind and spacing using the source file location information included in each token.

Finally, we implemented our language models using the PyTorch machine learning framework. To evaluate the models using our implementation, we divided corpus files into training, validation, and testing sets, and calculated the top-1 and top-3 accuracy of space prediction on the testing set after training. According to the results, which can be seen in Table 1, both the n-gram and the neural model are able to learn and suggest formatting conventions with high accuracy. However, the more sophisticated neural model performs significantly better than the n-gram model.

Table 1: Evaluation of Formatting Suggestions on our MathComp Corpus.

Model	Top-1 Acc.	Top-3 Acc.
Neural	**96.8%**	**99.7%**
n-gram	93.4%	98.9%

Challenges and Future Directions

Despite the high accuracy achieved by our preliminary implementation even when using the baseline n-gram model, we believe our spacing prediction (based only on raw token streams) needs significant tuning for practical use. For example, newlines before `Qed` sentences often get mispredicted, and unlike for name suggestions [3], it is usually inconvenient to inspect more than the top-1 suggestion for spacing. Moreover, for MathComp, we were able to construct, with help from maintainers, a sufficiently large corpus with strict adherence to conventions; for other projects, it may be more challenging, e.g., due to project size or lack of consensus on conventions. We ask the Coq community for input on the following challenges and directions:

Measuring successful predictions: Certain formatting errors, such as improper mid-sentence line breaks, are usually considered worse than others. Can we collaboratively define Coq-specific measures of formatting prediction success and use them to improve the models?

Finding conventions and corpus code: Which files in which projects are idiomatically formatted? What are the main coding styles used in the Coq community? With agreement on these questions, style conformity for files and whole projects can be precisely measured.

Manually improving generated suggestions: How do we best represent and apply rule-based conventions to do reranking of suggestions generated by a trained language model? How should we weigh manually specified conventions against learned ones?

Refactoring of code to adhere to conventions: Our preliminary implementation only modifies spacing, but code may require refactoring to properly address convention requirements, most simply, introducing or changing bullets and specific notations.

Integrating suggestions into the development process: How do we best provide our tools for suggesting conventions to the community? For example, displaying formatting suggestions during code reviews of pull requests on GitHub may work well for large projects, but small projects may have different workflows and thus benefit more from integration with IDEs.

References

[1] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, 51(4):81, 2018.

[2] E. J. Gallego Arias. SerAPI: Machine-friendly, data-centric serialization for Coq. Technical report, MINES ParisTech, 2016. https://hal-mines-paristech.archives-ouvertes.fr/hal-01384408.

[3] P. Nie, K. Palmskog, J. J. Li, and M. Gligoric. Deep generation of Coq lemma names using elaborated terms. In *IJCAR*, 2020. To appear. Extended version at https://arxiv.org/abs/2004.07761.

[4] M. E. Peters, W. Ammar, C. Bhagavatula, and R. Power. Semi-supervised sequence tagging with bidirectional language models. In *ACL*, pages 1756–1765, 2017.