Coqatoo: Generating Natural Language Versions of Coq Proofs

Andrew Bedford
Université Laval
Quebec, Canada
andrew.bedford.1@ulaval.ca

Abstract

We present Coqatoo, a command-line utility capable of automatically generating natural language versions of Coq proofs. We illustrate its use on a simple proof.

1 Introduction

Due to their numerous advantages, formal proofs and proof assistants, such as Coq, are becoming increasingly popular. One disadvantage of using proof assistants is that the resulting proofs can sometimes be hard to read and understand, particularly for less-experienced users. In an attempt to address this issue, Coscoy et al. [3] developed in 1995 an algorithm capable of *generating natural language proofs* from Coq's proof objects (i.e., calculus of inductive construction λ-terms) and implemented their approach in two development environments: CtCoq [4] and its successor Pcoq [5]. Unfortunately, these development environments are no longer available or maintained; Pcoq's last version dates from 2003 and requires Coq 7.4.

In order to bring this useful feature to modern development environments, we have implemented our own rewriting algorithm: Coqatoo.

2 Overview of Coqatoo

Much like Nuprl's text generation algorithm [6], Coqatoo uses high-level proof scripts to generate natural language proofs instead of the low-level proof objects used by Coscoy et al. By doing so, we can avoid the verbosity that comes from using low-level objects [2] and avoid losing valuable information such as the tactics that are used, the user's comments and the variable names.

Coqatoo's rewriting algorithm can be decomposed in two steps: information extraction and tactic-based rewriting.

Step 1: Information extraction Using an instance of the `coqtop` process and the proof script given as input, Coqatoo executes the tactics one by one and captures the intermediary proof states.

For example, Listing 1 represents the initial state of Listing 3's proof and Listing 2 represents the state after executing the first `intros` tactic.

```
1 subgoal

============================
forall P Q R : Prop, (P /\ Q -> R) <-> (P -> Q
  -> R)
```

Listing 1. State before executing the first intros tactic

```
1 subgoal

P, Q, R : Prop
============================
(P /\ Q -> R) <-> (P -> Q -> R)
```

Listing 2. State after executing the first intros tactic

These intermediary states, which contains the current assumptions and remaining goals, allow us to identify the changes caused by the execution of a tactic (e.g., added/removed variables, hypotheses or subgoals).

Step 2: Tactic-based rewriting Once the extraction of information is complete, we start the natural language proof generation. For each supported tactic, we have defined simple rewriting rules.

For example, for the `intros` tactic we first determine the types of the objects that are introduced. If they are variables, then we produce a sentence of the form `"Assume that ... are arbitrary objects of type ..."`. If they are hypotheses, then we instead produce a sentence of the form `"Suppose that ... are true"`. Finally, we insert a sentence indicating what is left to prove: `"Let us show that ..."`.

Note that the sentences that we use to produce the natural language versions are kept in files that are separate from the code. This allows Coqatoo to support multiple languages and proof styles. For the moment, it can output proofs in English or French, in plain text or in annotation mode (see Listing 4 for example). In annotation mode, each tactic is accompanied with an informal explanation. We believe that this format will be particularly useful for new Coq users.

2.1 Example

To illustrate our approach, consider the proof script in Listing 3.

```
Lemma conj_imp_equiv : forall P Q R:Prop,
  ((P /\ Q -> R) <-> (P -> Q -> R)).
Proof.
  intros.
  split.
  - intros H HP HQ.
    apply H.
    apply conj.
    -- assumption.
    -- assumption.
  - intros H HPQ.
    inversion HPQ.
    apply H.
    -- assumption.
    -- assumption.
Qed.
```

Listing 3. Proof script given as input

CoqPL 2018, Los Angeles, CA, USA
2016. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

```
Lemma conj_imp_equiv : forall P Q R:Prop, ((P /\ Q -> R) <-> (P -> Q -> R)).
Proof.
(* Assume that P, Q and R are arbitrary objects of type Prop. Let us show that (P /\ Q -> R) <-> (P ->
  Q -> R) is true. *) intros.
split.
  - (* Case (P /\ Q -> R) -> P -> Q -> R: *)
    (* Suppose that P, Q and P /\ Q -> R are true. Let us show that R is true. *) intros H HP HQ.
    (* By our hypothesis P /\ Q -> R, we know that R is true if P /\ Q  is true. *) apply H.
    apply conj.
    -- (* Case P: *)
       (* True, because it is one of our assumptions. *) assumption.
    -- (* Case Q: *)
       (* True, because it is one of our assumptions. *) assumption.
  - (* Case (P -> Q -> R) -> P /\ Q -> R: *)
    (* Suppose that P /\ Q and P -> Q -> R are true. Let us show that R is true. *) intros H HPQ.
    (* By inversion on P /\ Q, we know that P, Q are also true. *) inversion HPQ.
    (* By our hypothesis P -> Q -> R, we know that R is true if P -> Q  is true. *) apply H.
    -- (* Case P: *)
       (* True, because it is one of our assumptions. *) assumption.
    -- (* Case Q: *)
       (* True, because it is one of our assumptions. *) assumption.
Qed.
```

Listing 4. Output in annotation mode

3 Future Work

Coqatoo is only a proof-of-concept for the moment. As such, there remains much to be done before it can be of real use.

Increase the number of supported tactics The number of tactics that it supports is limited to only a handful (see Coqatoo's GitHub repository [1] for more details). We expect that, with the help of the community, we will be able to support enough tactics to generate natural language versions of most proofs in the *Software Foundations* book.

Add partial support for automation In regards to automation, Coqatoo only supports the `auto` tactic: if the `auto` tactic is present within the script, it is replaced with `info_auto` in order to obtain the sequence of tactics that is used by `auto`. We plan on adding partial support for automation in the future, starting with the chaining operator "`;`". To support this operator we will need to construct a tree representation of the proof. We are exploring the possibility of using the Prooftree library [7].

Automatically structure proofs We currently assume that proofs are structured using "-" bullets and use these to determine the level of indentation that must be added to each line. Using the proof tree, we should be able to automatically insert bullets and hence, determine the correct indentation.

Integration with development environments Once it is sufficiently developed, we plan on integrating our utility in modern Coq development environments such as CoqIDE and ProofGeneral.

Acknowledgments

We would like to thank Josée Desharnais, Nadia Tawbi, Souad El Hatib and the reviewers for their comments.

We would also like to thank the Coq community for the large number of resources and tutorials that are available online.

References

[1] Andrew Bedford. 2017. Coqatoo's Repository. https://github.com/andrew-bedford/coqatoo. (2017).
[2] Yann Coscoy. 1996. A Natural Language Explanation for Formal Proofs. In *Logical Aspects of Computational Linguistics, First International Conference, LACL '96, Nancy, France, September 23-25, 1996, Selected Papers*. 149–167. https://doi.org/10.1007/BFb0052156
[3] Yann Coscoy, Gilles Kahn, and Laurent Théry. 1995. Extracting Text from Proofs. In *Typed Lambda Calculi and Applications, Second International Conference on Typed Lambda Calculi and Applications, TLCA '95, Edinburgh, UK, April 10-12, 1995, Proceedings*. 109–123. https://doi.org/10.1007/BFb0014048
[4] Yves Bertot et al. 1997. CtCoq. https://www-sop.inria.fr/croap/ctcoq/ctcoq-eng.html. (1997).
[5] Yves Bertot et al. 2003. Pcoq. http://www-sop.inria.fr/lemme/pcoq/. (2003).
[6] Amanda M. Holland-Minkley, Regina Barzilay, and Robert L. Constable. 1999. Verbalization of High-Level Formal Proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and Eleventh Conference on Innovative Applications of Artificial Intelligence, July 18-22, 1999, Orlando, Florida, USA*. 277–284. http://www.aaai.org/Library/AAAI/1999/aaai99-041.php
[7] Hendrik Tews. 2017. Prooftree. https://askra.de/software/prooftree/. (2017).