Towards Automatic Transformations of Coq Proof Scripts

Nicolas Magaud

Towards Automatic Transformations of Coq Proof Scripts

Nicolas Magaud[1][0000−0002−9477−4394]

Lab. ICube UMR 7357 CNRS Université de Strasbourg, France
magaud@unistra.fr

Abstract. Proof assistants like Coq are increasingly popular to help mathematicians carry out proofs of the results they conjecture. However, formal proofs remain highly technical and are especially difficult to reuse. In this paper, we present a framework to carry out *a posteriori* script transformations. These transformations are meant to be applied as an automated post-processing step, once the proof has been completed. As an example, we present a transformation which takes an arbitrary large proof script and produces an equivalent single-line proof script, which can be executed by Coq in one single step. Other applications, such as fully expanding a proof script (for debugging purposes), removing all named hypotheses, etc. could be developed within this framework. We apply our tool to various Coq proof scripts, including some from the GeoCoq library.

Keywords: Coq, proof transformation, GeoCoq

1 Motivations

Proof assistants like Coq [1, 4] are increasingly popular to help mathematicians carry out proofs of the results they conjecture. However, formal proofs remain highly technical and are especially difficult to reuse. Once the proof effort is done, the proof scripts are left as they are and they often break when upgrading to a more recent version of the prover. To reduce the burden of maintaining the proof scripts of Coq, we propose a tool to post-process the proof scripts to make them cleaner and easier to reuse. The first transformation that we focused on consists in compacting a several-step proof script into a single-step proof script. Even though our framework can be used to implement other proof script transformations, this one is of special interest to us. Indeed, we recently designed a prover for projective incidence geometry [3, 12] which relies on the concept of rank to carry out proofs of geometric theorems such as Desargues or Dandelin-Gallucci automatically. This prover produces a trace (a large Coq proof script containing several statements and their proofs). We hope to use the proof transformation tool to shape up the automatically generated proofs and make them easier to reuse and integrate in larger proof repositories.

More generally, proof maintenance and reuse tools have been studied extensively by Talia Ringer et al. [11, 10]. Contrary to our approach, their tools aim

at fixing the issues when they occur. In our setting, we think it is better to try and improve the proof scripts so that they are less likely to break, even after several years and numerous updates of the components.

Outline of the paper The paper is organized as follows. In Sect. 2, we present a simple example of a proof script transformation. In Sect. 3, we describe the implementation of our tool as well as the future extensions we currently develop. In Sect. 4, we present some concluding remarks and the perspectives of this work.

2 Transforming Large Proof Scripts into One-line Scripts

The Coq tactic language [5] features some tacticals to execute some tactics in a sequence `tac1;tac2;tac3` or try and execute different tactics on the same goal `solve [ tac1 | tac2 | tac3]`. Moreover these tacticals can be combined. E.g. `tac0 ; [tac1 | tac2 | tac3]` runs the first tactic `tac0` which should yield 3 subgoals. The first one is solved using `tac1`, the second one using `tac2` and the third one using `tac3`. Once a proof script is written (as several steps) by the user, we can use these tacticals to build an equivalent proof script, which can be executed in a single step.

Let us consider a simple example, proving the distributivity of the connective or (`\/`) over the connective and (`/\`) as shown in the statement of Fig. 1.

```
Lemma foo : forall A B C : Prop, A \/ (B /\ C) -> (A\/B)/\(A\/C).
```

```
Proof.
  intros; destruct H.
  split.
  left; assumption.
  left; assumption.
  destruct H.
  split.
  right; assumption.
  right; assumption.
  Qed.
```

```
Proof.
intros; destruct H;
  [ split;
    [ left; assumption
    | left; assumption ]
  | destruct H ;
    split;
    [ right; assumption
    | right; assumption ] ].
Qed.
```

Fig. 1. A user-written script (left) and the equivalent single-step script (right)

The left-hand side presents the proof script that one may expect from a master student, factorizing some parts but still decomposing the reasoning in several steps. On the right-hand side, we propose a one-line script to carry out exactly the same proof.

In Coq, writing directly the right-hand side is almost impossible, whereas it is fairly easy to generate it automatically from the left-hand side. In the Coq standard library, several lemmas are proved using a single one-line tactic. The

main advantage is that it provides concise and structured proofs but it has the drawback that, when something goes wrong, it is hard to debug and fix it.

3 Experiments, Limitations and Results

3.1 Implementation

We choose to implement our tool in `OCaml`, using the serialisation mechanism `serapi` [6] developed by Emilio Gallego Arias for communication with the Coq proof assistant. Our tool uses anonymous pipes to communicate with `serapi`, which itself sends requests to Coq and retrieves the answers. Commands are kept as in the input file. Tactics are aggregated using tacticals such as `;`, `[` and `]`. At each step of the proof, we compare the current number of subgoals to the number of subgoals right before the execution of the current tactic. If it is the same, we simply concatenate the tactics with a `;` between them. If the number of goals increases, we open a square bracket `[` and push into the stack the previous number of goals. Each time a goal is solved, we check whether some goals remain to be proved at this level. If yes, we add another `;` and then focus on the next subgoal. If there are no more subgoals at this level, we pop the 0 from the top of the stack, thus closing the current level with a `]` and carry on with subgoals of the previous level.

The source code[1] as well as some examples are freely available online. It is developped using `Coq` 8.17.0 and the corresponding `serapi` version 8.17.0+0.17.0.

3.2 Limitations

So far, commands and tactics are told apart simply by assuming commands start with a capital letter [A-Z] and tactics with a small letter [a-z]. This convention is well-known in Coq, however in some developments (e.g. GeoCoq), some ad-hoc user tactics may start with a capital letter. Handling this properly requires additional developments and is currently under way.

To make the transformation easier, a first phase could be added to our proof script transformer to remove all commands which lay among the proof steps (e.g. `Check`, `Print` or `Locate`) and make sure all tactics names start with a small letter.

Finally, Coq proof scripts can be structured using bullets (`+`, `-`, `*`) as well as curly brackets to identify some subproofs. In addition, one can direct work on a goal which is not the current one using the `2:tac.` notation which performs the tactic `tac` on the second goal of the subgoals. We still need to devise a way to deal properly with such partially-structured proof script.

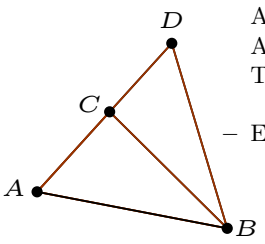
[1] `https://github.com/magaud/coq-lint`

3.3 Successful Transformations

In addition to our test suite examples, we consider more challenging proof scripts. We successfully transformed a library file from the Standard Library of Coq: Cantor.v[2] from the `Arith` library as well as some large files from the `GeoCoq` library [2, 8] (e.g. `orthocenter.v`[3]). As the tool gets more mature, we plan to transform more files, and we shall especially focus on the `GeoCoq` library which features several different proof styles and thus shall allow us to evaluate the robustness of our tool.

3.4 Refactoring Proof Scripts Automatically Generated by our Prover for Projective Incidence Geometry

We recently developed a new way [3, 12], based on ranks, to automatically prove statements in projective incidence geometry. Our approach works well but produces proof scripts which are very large and often feature several auxiliary lemmas. Fig. 2 presents a very simple example `LABC`, which is formally proven by our tool but yields a fairly verbose proof script using one intermediate lemma `LABCD` (see appendix A for details).



– Informal statement
Assume that ABD is a triangle,
Assume that C is a point on AD, such that C≠ A and C≠ D,
Then ABC is a triangle.

– Expressed using ranks

$\forall A, B, C, D$: `Point`,
$rk\{A, D, B\} = 3 \rightarrow rk\{A, C, D\} = 2 \rightarrow$
$rk\{C, A\} = 2 \rightarrow rk\{C, D\} = 2 \rightarrow$
$rk\{A, C, B\} = 3.$

Fig. 2. An example of a statement in projective geometry, formalized using ranks

We plan to use our script transformation tool to refactor automatically generated proof scripts, inlining auxiliary lemmas and thus making proof scripts more concise and hopefully more readable for humans.

3.5 Next steps

To fully evaluate the tool, we need to handle larger examples, outside of the standard library of Coq. The next step consists in improving Coq options handling (e.g. `-R`) to our script transformation tool to tackle other formal proof libraries.

[2] `https://github.com/coq/coq/blob/master/theories/Arith/Cantor.v`
[3] `https://github.com/GeoCoq/GeoCoq/blob/master/Highschool/orthocenter.v`

We also plan to propose the reciprocal script transformation, turning a single-step proof script into a more detailed (easier to debug) proof script. This could especially be useful when porting formal proofs from one version of Coq to the next one.

Other applications of interest could be to remove the names of all variables or hypotheses from the scripts, or at least to force them to be explicitly introduced. The script snippet `intros; apply H` could be replaced by a more precise one `intros n p H; apply H`. This way, we could ensure that the proofs are not broken when the names of automatic variables change. From a reliability point of view, it would be even better to use the tactic `intros; assumption`. Although its cost is higher (because we need to search the correct hypothesis among all of them every time we run the tactic), it does not depend on some arbitrary variable names.

Finally, regarding our current implementation, it would be interesting to benchmark the transformation to see whether transforming the whole standard library of Coq into single-step proof scripts could improve the compilation time of this library.

4 Conclusions and perspectives

We build a proof script transformation tool, which transforms an arbitrary large proof script into a single-step *one-Coq-tactic* proof script. This tool has been successfully experimented on some significant library files from the Coq ecosystem.

This first example shows that the approach is sound and we plan to extend it to integrate tactic languages such as ssreflect [7], Ltac2 [9] or Mtac [13] in our framework. In the longer term, we expect to design some new proof script transformations and combine them in order to build more reliable proof developments which can last longer and would be easier to maintain. Among these transformations, we shall start with a mechanism to transform a proof script into a sequence of atomic proof steps (to make debugging easier when the proof breaks). We may also study how to transform proofs carried out automatically by their actual traces, avoiding recomputing the proof search each time the proof is re-runned.

References

1. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development, Coq'Art : The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, An EATCS Series, Springer-Verlag, Berlin/Heidelberg (May 2004), 469 pages
2. Boutry, P., Gries, C., Narboux, J., Schreck, P.: Parallel postulates and continuity axioms: a mechanized study in intuitionistic logic using Coq. Journal of Automated Reasoning p. 68 (2017). https://doi.org/10.1007/s10817-017-9422-8

3. Braun, D., Magaud, N., Schreck, P.: Two new ways to formally prove dandelin-gallucci's theorem. In: Chyzak, F., Labahn, G. (eds.) ISSAC '21: International Symposium on Symbolic and Algebraic Computation, Virtual Event, Russia, July 18-23, 2021. pp. 59–66. ACM (2021). https://doi.org/10.1145/3452143.3465550
4. Coq development team: The Coq Proof Assistant Reference Manual, Version 8.13.2. INRIA (2021), http://coq.inria.fr
5. Delahaye, D.: A Tactic Language for the System Coq. In: Parigot, M., Voronkov, A. (eds.) Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings. Lecture Notes in Computer Science, vol. 1955, pp. 85–95. Springer (2000). https://doi.org/10.1007/3-540-44404-1_7
6. Gallego Arias, E.J.: SerAPI: Machine-Friendly, Data-Centric Serialization for Coq. Tech. rep., MINES ParisTech (Oct 2016), https://hal-mines-paristech.archives-ouvertes.fr/hal-01384408
7. Gonthier, G., Mahboubi, A.: An introduction to small scale reflection in Coq. J. Formaliz. Reason. **3**(2), 95–152 (2010). https://doi.org/10.6092/issn.1972-5787/1979
8. Narboux, J.: Mechanical Theorem Proving in Tarski's geometry. In: Eugenio Roanes Lozano, F.B. (ed.) Automated Deduction in Geometry 2006. LNCS, vol. 4869, pp. 139–156. Francisco Botana, Springer, Pontevedra, Spain (Aug 2006). https://doi.org/10.1007/978-3-540-77356-6
9. Pédrot, P.M.: Ltac2: Tactical Warfare. In: Krebbers, R., Sergey, I. (eds.) Proceedings of the CoqPL workshop 2019 (2019)
10. Ringer, T., Palmskog, K., Sergey, I., Gligoric, M., Tatlock, Z.: QED at large: A survey of engineering of formally verified software. Found. Trends Program. Lang. **5**(2-3), 102–281 (2019). https://doi.org/10.1561/2500000045
11. Ringer, T., Yazdani, N., Leo, J., Grossman, D.: Adapting proof automation to adapt proofs. In: Andronick, J., Felty, A.P. (eds.) Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2018, Los Angeles, CA, USA, January 8-9, 2018. pp. 115–129. ACM (2018). https://doi.org/10.1145/3167094
12. Schreck, P., Magaud, N., Braun, D.: Mechanization of incidence projective geometry in higher dimensions, a combinatorial approach. In: Janicic, P., Kovács, Z. (eds.) Proceedings of the 13th International Conference on Automated Deduction in Geometry, ADG 2021, Hagenberg, Austria/virtual, September 15-17, 2021. EPTCS, vol. 352, pp. 77–90 (2021). https://doi.org/10.4204/EPTCS.352.8
13. Ziliani, B., Dreyer, D., Krishnaswami, N.R., Nanevski, A., Vafeiadis, V.: Mtac: A monad for typed tactic programming in coq. J. Funct. Program. **25** (2015). https://doi.org/10.1017/S0956796815000118

A Proof Script for our basic example

```
Lemma LABCD : forall A B C D ,
rk(A:: C::nil) = 2 -> rk(A:: B:: D::nil) = 3 ->
rk(C:: D::nil) = 2 -> rk(A:: C:: D::nil) = 2 ->
rk(A:: B:: C:: D::nil) = 3.
Proof.
intros A B C D
```

```
HACeq HABDeq HCDeq HACDeq .
assert(HABCDm2 : rk(A:: B:: C:: D:: nil) >= 2).
{
assert(HACmtmp : rk(A:: C:: nil) >= 2)
       by (solve_hyps_min HACeq HACm2).
assert(Hcomp : 2 <= 2) by (repeat constructor).
assert(Hincl : incl (A:: C:: nil) (A:: B:: C:: D:: nil))
       by (repeat clear_all_rk;my_in0).
apply (rule_5 (A:: C:: nil) (A:: B:: C:: D:: nil) 2 2 HACmtmp Hcomp Hincl).
}
assert(HABCDm3 : rk(A:: B:: C:: D:: nil) >= 3).
{
assert(HABDmtmp : rk(A:: B:: D:: nil) >= 3)
       by (solve_hyps_min HABDeq HABDm3).
assert(Hcomp : 3 <= 3)
       by (repeat constructor).
assert(Hincl : incl (A:: B:: D:: nil) (A:: B:: C:: D:: nil))
       by (repeat clear_all_rk;my_in0).
apply (
  rule_5 (A:: B:: D:: nil) (A:: B:: C:: D:: nil) 3 3 HABDmtmp Hcomp Hincl
      ).
}
assert(HABCDM : rk(A:: B:: C:: D::nil) <= 3)
       by (solve_hyps_max HABCDeq HABCDM3).
assert(HABCDm : rk(A:: B:: C:: D::nil) >= 1)
       by (solve_hyps_min HABCDeq HABCDm1).
intuition.
Qed.

Lemma LABC : forall A B C D ,
rk(A:: C::nil) = 2 -> rk(A:: B:: D::nil) = 3 ->
rk(C:: D::nil) = 2 -> rk(A:: C:: D::nil) = 2 ->
rk(A:: B:: C::nil) = 3.
Proof.
intros A B C D
HACeq HABDeq HCDeq HACDeq .

assert(HABCm2 : rk(A:: B:: C:: nil) >= 2).
{
assert(HACmtmp : rk(A:: C:: nil) >= 2)
      by (solve_hyps_min HACeq HACm2).
assert(Hcomp : 2 <= 2)
     by (repeat constructor).
assert(Hincl : incl (A:: C:: nil) (A:: B:: C:: nil))
     by (repeat clear_all_rk;my_in0).
```

```
apply (
  rule_5 (A:: C:: nil) (A:: B:: C:: nil) 2 2 HACmtmp Hcomp Hincl
      ).
}
assert(HABCm3 : rk(A:: B:: C:: nil) >= 3).
{
assert(HACDMtmp : rk(A:: C:: D:: nil) <= 2)
       by (solve_hyps_max HACDeq HACDM2).
assert(HABCDeq : rk(A:: B:: C:: D:: nil) = 3)
       by
       (apply LABCD with (A := A) (B := B) (C := C) (D := D) ; assumption).
assert(HABCDmtmp : rk(A:: B:: C:: D:: nil) >= 3)
       by (solve_hyps_min HABCDeq HABCDm3).
assert(HACmtmp : rk(A:: C:: nil) >= 2)
       by (solve_hyps_min HACeq HACm2).
assert(  Hincl :
         incl (A:: C:: nil)
         (list_inter (A:: B:: C:: nil) (A:: C:: D:: nil)))
       by (repeat clear_all_rk;my_in0).
assert(  HT1 :
         equivlist (A:: B:: C:: D:: nil)
         (A:: B:: C:: A:: C:: D:: nil))
       by (clear_all_rk;my_in0).
assert( HT2 :
        equivlist (A:: B:: C:: A:: C:: D:: nil)
        ((A:: B:: C:: nil) ++ (A:: C:: D:: nil))
      ) by (clear_all_rk;my_in0).
rewrite HT1 in HABCDmtmp;rewrite HT2 in HABCDmtmp.
apply (
  rule_2
     (A:: B:: C:: nil) (A:: C:: D:: nil) (A:: C:: nil)
     3 2 2 HABCDmtmp HACmtmp HACDMtmp Hincl
     ).
}
assert(HABCM : rk(A:: B:: C::nil) <= 3)
       by (solve_hyps_max HABCeq HABCM3).
assert(HABCm : rk(A:: B:: C::nil) >= 1)
       by (solve_hyps_min HABCeq HABCm1).
intuition.
Qed.
```