Post-processing Coq Proof Scripts to Make Them More Robust

Titouan Lozac'h [1] and Nicolas Magaud
Lab. ICube UMR 7357 CNRS Université de Strasbourg

EuroProofNet 2024
2nd Workshop on the development, maintenance, refactoring and search of large libraries of proofs
Tbilissi, September 13-14, 2024

1. L3 student at ENS Paris-Saclay, summer intern at ICube

Outline

Motivations

- Proof assistants like Coq are increasingly popular.
- 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.
- Our goal : setting up some preventive maintenance tools to make porting proofs easier in the future.
- Possible transformations :
  - Adding structure to proof scripts
  - Removing explicit variables names
  - Inlining auxiliary lemmas
  - Decomposing a proof script into atomic steps (debug)
  - etc.

Outline

Coq tactic language

- Basic tactics : intros, apply, elim, induction, split, lia, nia
- Tacticals (to combine tactics in different ways) :
  - tac1 ; tac2
  - solve [tac1 | tac2 | tac3]
  - first [tac1 | tac2 | tac3]
  - ...
- We can transform any proof script into an equivalent single-step proof script.
- Example : distributivity of or (\/) over and (/\)

A user-written script and the equivalent single-step script

```
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.
```

The Inverse Transformation

- Compact proof scripts are :
  - nice for libraries (esp. for compiling them),
  - but painful for debugging.
- Hence, we implement the inverse transformation : fulling expanding and structuring proof scripts.

Back to our Example

```
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.
```

Outline

Implementation

- Prototype : independent from Coq, implemented in OCaml
- Uses the serialisation mechanism serapi (E. Gallego Arias) for communication with Coq.
- Inter-processes communication using anonymous pipes
- Commands and comments are kept as they are.
- Tactics are aggregated using the tacticals `;`, `[` and `]`.

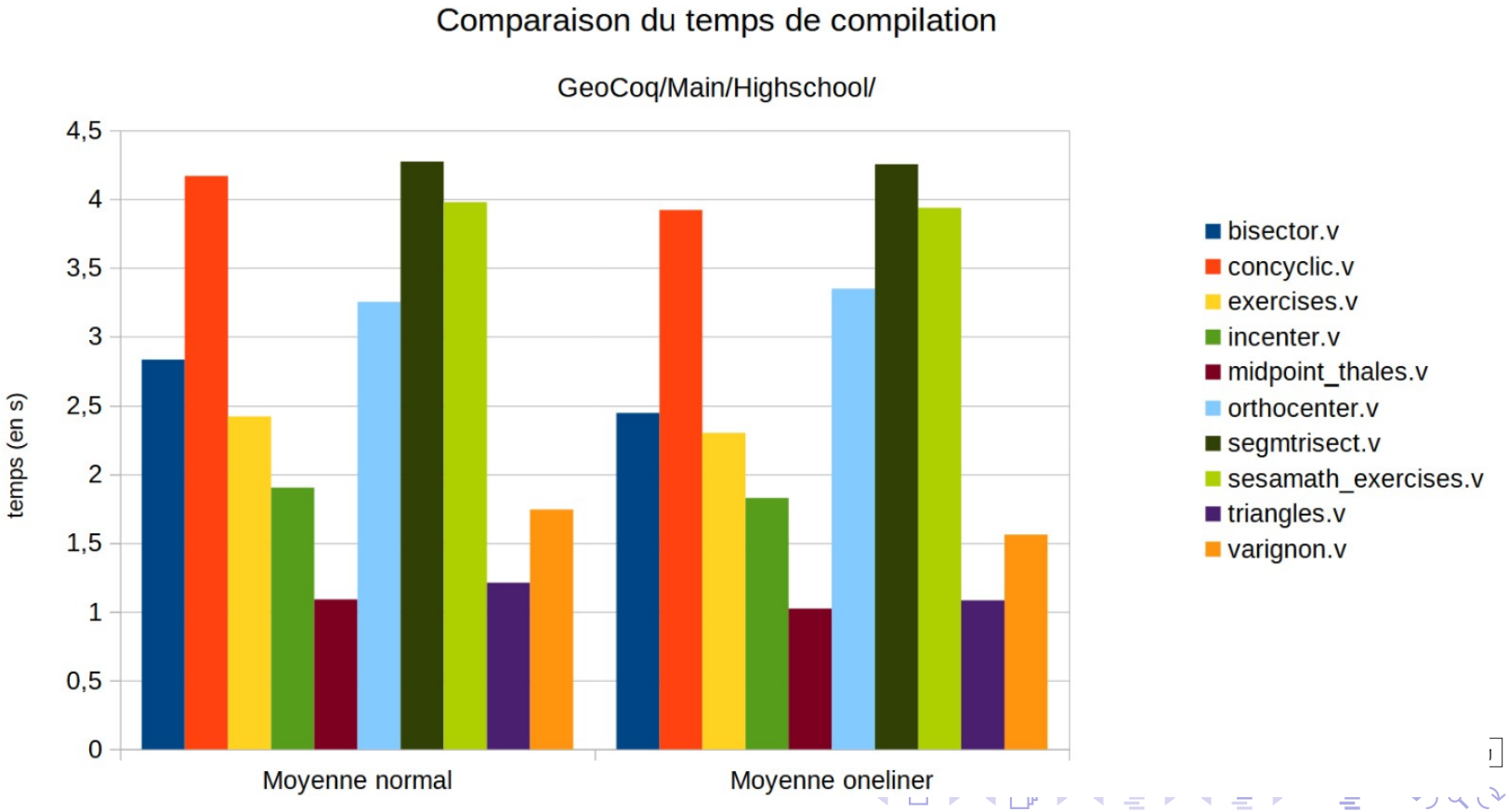
Outline of the implementation

- Building one-line proof scripts
  - Loading the whole file
  - Executing it at a whole until the end
  - Querying statements and proof goals
  - Structuring according to the change of numbers of goals

    `== ;` `+1 [` `<>0 |` `==0 ]`
- Fully expanding a one-line proof scripts
  - Similar approach
  - The proof structure is provided using bullet points
  - Tricky point : t1 ; t2.
    t1 ; t2. means t1. t2. or t1 ; [t2 | t2]. or t1 ; [t2 | t2 | t2]., etc.
  - Therefore executing the script has to be done step by step with possible rollbacks.

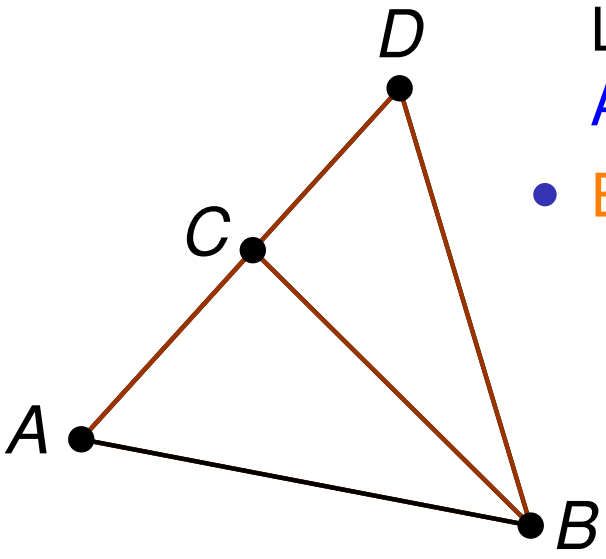
Some Successful Transformations

- Examples : files from the Arith library of Coq and from the Highschool library of GeoCoq
- Transformations achieved in both directions
- One-step proof scripts improves compilation time by 5%



Outline

Next stage : integrating an automated prover for geometry in Coq



- A simple example
  Let ABD be a triangle,
  Let C be a point on AD, C$\neq$ A and C$\neq$ D
  ABC is a triangle
- Expressed using ranks

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

Next Stage : Refactoring proof scripts

- Our automated prover for projective geometry [2] generates Coq proof scripts
- Proof scripts are large, verbose, but easy to debug
- Integrating it into Coq requires simpler proof scripts without auxiliary lemmas.
- We propose a three-step process :
  - first generating the proof,
  - then inlining the lemmas (such as `LABCD` in the example),
  - finally making it a one-line proof script.

2. Braun, Magaud, Schreck - ADG2021/JAR2024

An Example (I)

```
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.
```

An Example (II)

```
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 :
[ ... 24 more lines ... ]
Qed.
```

Outline

Conclusions and Perspectives

- Achievements
  - `coq-lint` : a proof script transformation tool
  - allows refactoring of proof scripts into *one-Coq-tactic* proofs
  - allows adding structure to proof scripts
- Future Work
  - Remove some specific tactics
  - Transform automated proofs by their actual traces
  - Inline some lemma applications into the body of the proofs
  - Make introduced variables all explicit or all implicit, . . .

Thanks ! Questions ?

https://github.com/magaud/coq-lint
https://gitlab.crans.org/titloz/stage13