Automatic Transformations of Coq Proof Scripts

Nicolas Magaud
Lab. ICube UMR 7357 CNRS Université de Strasbourg

ADG 2023
International Workshop on Automated Deduction in Geometry
Belgrade, September 20-22, 2023

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

1. Motivations
2. Transforming Large Proof Scripts into One-line Scripts
3. Experiments, Limitations and Results
4. Conclusions and Perspectives

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.
```

Outline

Implementation

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

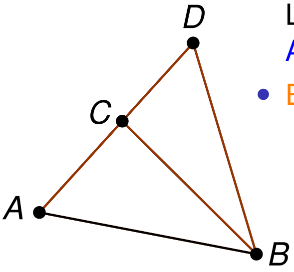
Outline of the implementation

- 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.

Some Successful Transformations

- Several simple benchmark examples
- Example files from the Coq Std Library (Arith) : e.g. Cantor.v (88 lines)
- A file from the GeoCoq library : orthocenter.v (329 lines), more to come. . .

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 (Braun, Magaud, Schreck - ADG2021) 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 two-step process :
  - first generating the proof,
  - and then shrinking it.

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_inO).
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_inO).
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_inO).
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)))
```

Outline

Conclusions and Perspectives

- Achievements
  - `coq-lint` : a proof script transformation tool
  - builds *one-Coq-tactic* proofs
- Future Work
  - Decompose a proof into a sequence of atomic proof steps
  - 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`