Post-processing of Rocq Proof Scripts

Alexandre Jean and Nicolas Magaud
Lab. ICube UMR 7357 CNRS Université de Strasbourg

EuroProofNet Workshop on Proof Libraries
Orsay, September 15, 2025

Outline

Motivations

- Proof assistants like Rocq 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
  - Replacing call to auto/ltac tactics with the actual proof steps
  - Making all variables names implicit or explicit
  - Inlining auxiliary lemmas
  - Decomposing a proof script into atomic steps (debug)
  - etc.

Rocq 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]
  - ...
- Advanced tactics : auto, intuition
- A first example : transforming a 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. to compile them efficiently),
  - but painful for debugging.
- Hence, we also implement the inverse transformation : fulling expanding and structuring proof scripts.

Back to our Example

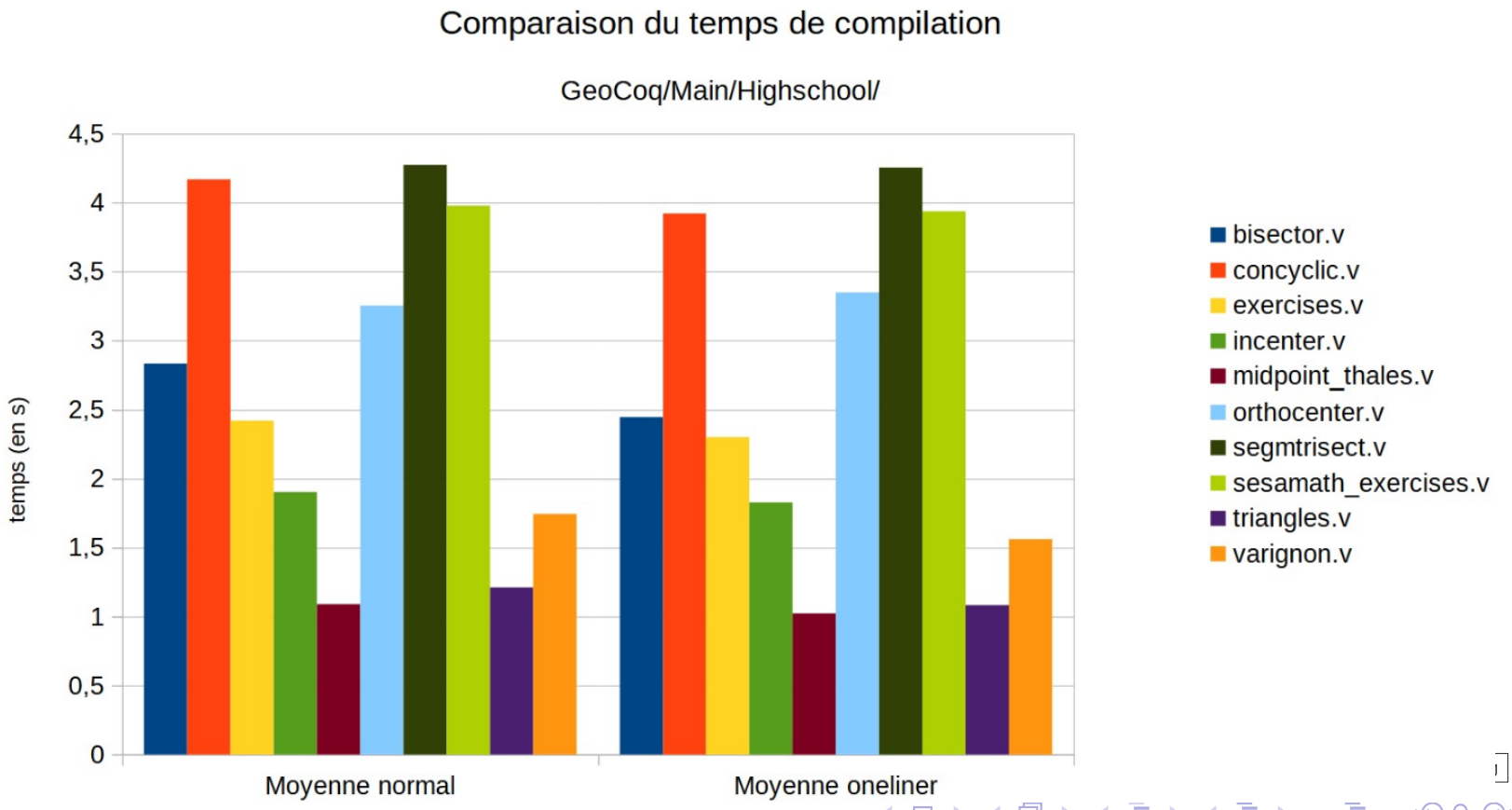
```
Lemma foo : forall A B C : Prop,
            A \/ (B /\ C) -> (A\/B)/\(A\/C).
```

```
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.
```

Some Results

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



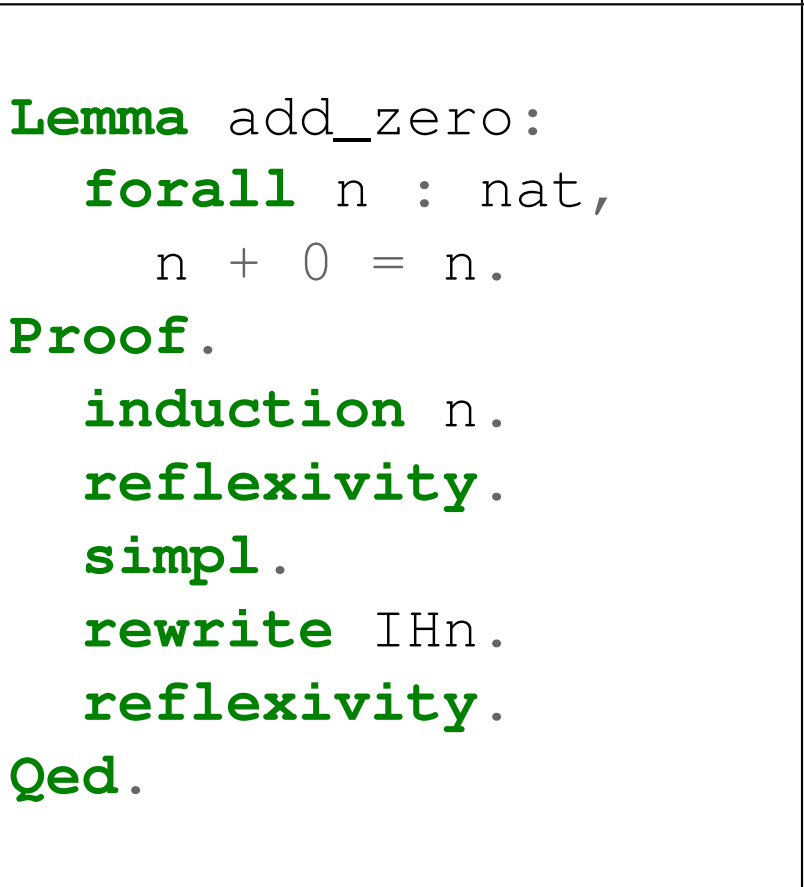
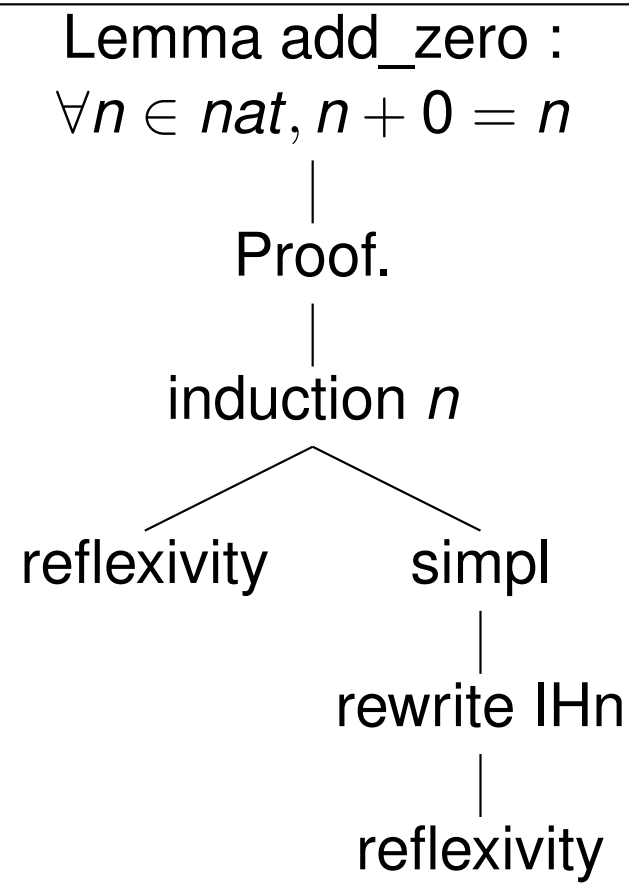
Outline

Rocq-ditto

- An external tool to perform source-to-source transformations of Rocq proof scripts
- Implemented as an Ocaml library handling Rocq AST [1]
- Uses *rocq-lsp* to get a Rocq AST from a file
- Allows for easy Rocq-AST rewriting by automatically moving other AST nodes when adding, removing or replacing a node
- Dual representation of proofs : proof-tree and linear structure
- Allows for speculative execution
- Provides quoting and unquoting functions

1. Abstract Syntax Trees

Internal Representation of Proof Scripts

Rocq proof script	associated Rocq-ditto proof tree
	


```
Lemma add_zero:
  forall n : nat,
    n + 0 = n.
Proof.
  induction n.
  reflexivity.
  simpl.
  rewrite IHn.
  reflexivity.
Qed.
```

How to Define a Transformation with Rocq-ditto

Transformation : A transformation is a function f that takes a proof as input and returns a list of transformation steps drawn from the set

$$\{Remove(id),\ Replace(id, new\_node),\ Add(new\_node),\ Attach(new\_node, attach\_position, anchor\_id)\}$$

- **Remove**(*id*) : remove the node identified by *id*.
- **Replace**(*id*, *new_node*) : replace the node identified by *id* with *new_node*
- **Add**(*new_node*) : add a new node to the AST
- **Attach**(*new_node*,*attach_position*,*anchor_id*) : places *new_node* on a position relative to the node with the id *anchor_id*.

Outline

Four Use Cases

- Structuring / compacting proof scripts
- Replacing auto calls by their actual proof steps
- Explicit naming of automatically introduced variables
- Constructivization of the GeoCoq library

Replacing auto calls by their computational contents

```
Lemma bar :  forall P Q R S : Prop,
    (P -> Q) -> (Q -> R) -> (R -> S) -> (P \/ S) ->
    (Q \/ R \/ S).
```

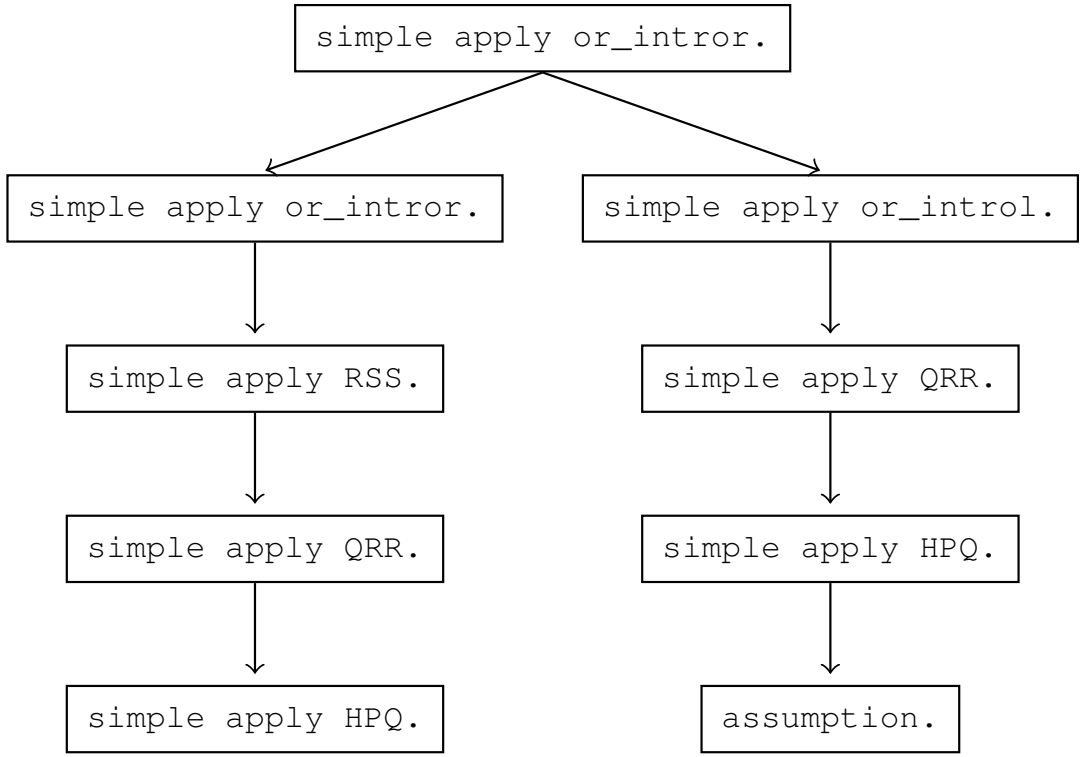
```
Proof.
 intros
   P Q R S HPQ QRR RSS H.
 destruct H.
 auto.
 right; right.
 assumption.
Qed.
```

```
Proof.
 intros
   P Q R S HPQ QRR RSS H.
 destruct H.
 simple apply or_intror.
 simple apply or_introl.
 simple apply QRR.
 simple apply HPQ.
 assumption.
 right; right.
 assumption.
Qed.
```

Using info_auto to retrieve the actual proof steps

- info_auto provides insights about what auto does.
- Using speculative execution, we can rebuild the actual proof steps.

```
simple apply or_intror.
 simple apply or_intror.
  simple apply RSS.
   simple apply QRR.
    simple apply HPQ.
 simple apply or_introl.
  simple apply QRR.
   simple apply HPQ.
    assumption.
```



Explicitly Naming all Variables

- Deals with all tactics generating new names (intros, inversion, induction, destruct, etc.)
- Transforms a fragile proof script

```
intros.
rewrite IHa.
```

- into a robust proof script

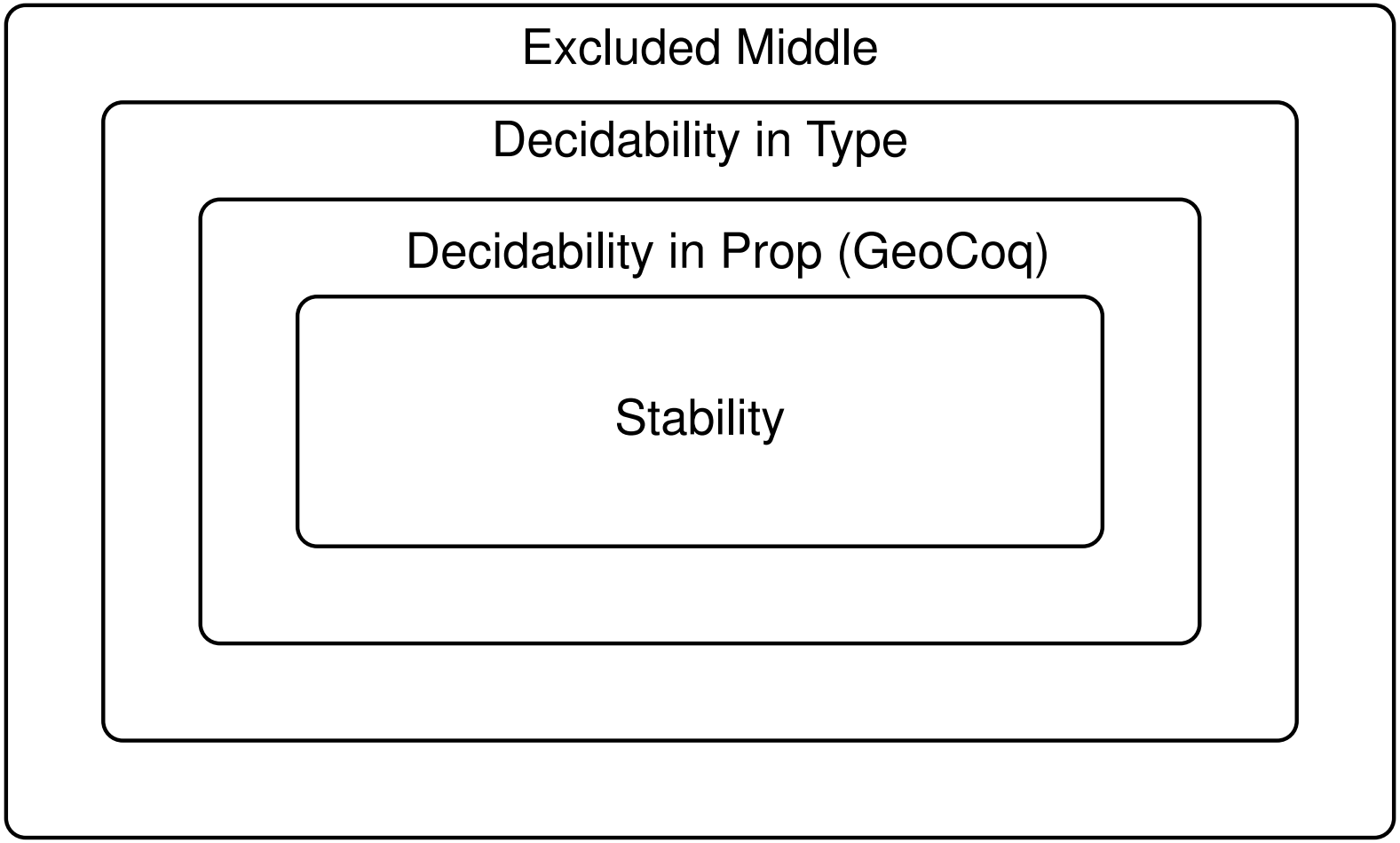
```
intros n m Hnm IHa.
rewrite IHa.
```

- As we assume the proof script compiles without errors, then the names are appropriate.
- The automatically assigned names are explicitly specified.

Constructivization of the GeoCoq library

- GeoCoq : a formal Rocq library, formalizing geometry including its arithmetization
- Based on Tarski axioms for geometry and decidability of point equality
- Contructivizing the arithmetization of geometry :
  - The arithmetization of geometry can be obtain without assuming any decidability property.
  - It relies on Beeson's main result in *A constructive version of Tarski's geometry*.

Decidability



Definition

The stability of an unary predicate P states

$$\forall x, \neg\neg P(x) \rightarrow P(x)$$

It is trivial to show that if an unary predicate P is decidable, then it is also stable.

Stability of Predicates

- Logical connectives

  We have : $\forall AB, \text{stable}\, A \implies \text{stable}\, B \implies \text{stable}\, A \wedge B$

  However, it does not hold $A \vee B$. We introduce a new negative formula, $\neg(\neg A \wedge \neg B)$, noted $A \sqcup B$, which preserves the stability of propositions.

- Stability of equality, congruence and betweenness
  - Stability of point equality : $\neg\neg X = Y \implies X = Y$
  - We deduce the stability of the congruence predicate Cong, but not of the betweenness predicate Bet, we could only prove its stability under a non-degeneracy assumption : $\forall ABC, A \neq B \implies \neg\neg\, \mathrm{Bet}\, ABC \implies \mathrm{Bet}\, ABC$

Using Rocq-ditto to Make Proof Scripts Constructive

- Useful transformations
  - One that admits proofs involving *exists* in the statement.
  - One that replaces usual predictates into stable ones.
  - One that replaces classical tactics like `left` with constructive alternatives, here `stab_left`.

```
Lemma by_left : forall A B : Prop,
A -> A \_/ B.
Proof. unfold or_dM; tauto. Qed.

Ltac stab_left :=
match goal with
| |- ?A \_/ ?B => apply (by_left A B)
end.
```

  - Still work in progress. Rocq-ditto is a nice helper to translate the GeoCoq library into a constructive one.

Outline

Combining Transformations on Proof Scripts

- What is a improved proof script ?
- Depends on the user, their individual needs
- More compilation-efficient ? more readable ? shorter ?
- Issues to be addressed :
  - Reversibility
  - Compositionality
  - Appropriate order of the transformations
  - Optimality issues ? w.r.t performance ? w.r.t. readability ?

Outline

Conclusions and Perspectives

- Achievements
  - A framework `rocq-ditto` to handle Rocq proof scripts
  - Allows refactoring of proof scripts in various ways (factorizing, adding structure, inlining, . . .)
  - Multi-criteria optimization (accomodating various proof styles, various purposes, etc.)
  - Implements some specific transformations to achieve the constructivization of the GeoCoq library
- Future Work
  - Removing all occurences of each named variable
  - Scaling the infrastructure to a whole library handler
  - More abstract data-structures to represent proof scripts ?
  - Integration to vscoq ?

Thanks ! Questions ?

https://github.com/blackbird1128/coq-ditto

[1] *Alexandre Jean*. A library for the automated transformation of Rocq AST. Rocqshop 2025, Reykjavik, Iceland, Sept. 2025.

[2] *Alexandre Jean, Pierre Boutry and Nicolas Magaud*. An Automated Approach towards Constructivizing the GeoCoq Library. Automated Deduction in Geometry (ADG). July 2025.

[3] *Alexandre Jean and Nicolas Magaud*. Transformations automatisées de preuves Coq. In Approches Formelles dans l'Assistance au Développement du Logiciel (AFADL), Pau, France, June 2025.