Université de Strasbourg

A library for the automated transformation of Rocq AST

Alexandre Jean

University of Strasbourg

27 september 2025

Context

- Proofs in proof assistant like **Rocq** are used to formalize mathematics and verify programs using proof script.
- Many of these proof scripts are **brittle** and difficult to maintain and reuse.
- How can we improve some criteria such as modularity, automatisation and robustness of Rocq proof ?

State of the art: transformations on Rocq code

We found multiple examples of transformations on Rocq code, each with their own implementation:

- *Towards Automatic Transformations of Coq Proof Scripts* [4]
- *Designing Proof Deautomation in Rocq* [5]
- *Post-processing Coq Proof Scripts to Make Them More Robust* [3]
- *Code Generation via Meta-programming in Dependently Typed Proof Assistants* [1]

Rocq-ditto

- Rocq AST[1] rewriting Ocaml library.
- source to source transformations.
- Use *rocq-lsp* [2] 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.
- Compatible with Ocaml standard library functions such as *filter*, *fold*, *map*.
- Dual representation of proof: proof-tree and linear structure.
- Allow for speculative execution.
- Quoting and unquoting functions.
- Compatible with modern Rocq 9.0.0 and previous version back to 8.17.

[1] Abstract Syntax Tree

How to define a transformation with Rocq-ditto

Definition

Transformation: A transformation is a function f that take a proof as input and return 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 $anchor\_id$.

Some current transformations

- A transformation to replace *auto* by the tactics computed by *auto* .
- A transformation to replace multiple consecutive call to *intro* by a single *intros* call.
- A transformation to remove unnecessary tactics in a proof.
- A transformation to replace call to tactics creating fresh variables such as *intros* with *intros* V_1 V_2 ... V_n where each V_i corresponds to a variable automatically introduced by the tactic.

Zoom on a Transformation: Replacing tactics introducing fresh variables automatically with a fixed version

```
Theorem fact_grow_weak :
 forall n m : ℕ,
 n < m ->  n! <= m!.
Proof.
  induction m.
  lia.
  simpl.
  intros.
  inversion H.
  lia.
  lia.
Qed.
```

Automatically introduced variables

```
Theorem fact_grow_weak_ :
 forall n m : ℕ,
 n < m ->  n! <= m!.
Proof.
  induction m as [|m IHm].
  lia.
  simpl.
  intros H.
  inversion H as [|m0 H1 H0].
  lia.
  lia.
Qed.
```

Fixed variables names

Using speculative execution to get the variables before and after a tactic execution

Let $r(\text{State}(S), \textit{tactic})$ be a pure function that take a state $\text{State}(S)$ and returns a new state $\text{State}(S_1)$ equal to the state after apply *tactic* to S.

Idea: For each tactic L_i in a proof $L = [L_1, \ldots, L_n]$, compute

- $\text{State}(L_i)$
- $\text{State}(L_{i+1})$

Method:

- Take the proof state after $L_1, \ldots, L_{i-1}$.
- Run L_i on that proof state to get the new state.
- Extract variable names before/after.

Zoom on a Transformation: Replacing *intros* with *intros* V_1 V_2 ... V_n

Identifying *intros* tactics:

- Quote each proof node and check if its string representation is "*intros*".
- *Intros* automatically introduces variables into the context.

Extracting new variables:

- For each node, track proof states before and after execution.
- V_{prev}: variables before; V_{intros}: variables after.
- New variables $= V_{\text{intros}} \setminus V_{\text{prev}}$.

Constructing new steps:

- Concatenate names of new variables to "intros".
- Quote into an AST node and wrap in a $Replace$ step.
- Final result: list of $Replace$ steps for each *intros* tactic.

Conclusion

Summary

- Goal : Improve the maintenance and robustness of Rocq proof automatically.
- Our solution : Automated transformation of Rocq proof defined by the user.
- Results : A library allowing users to write transformations and some simple transformations examples.

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

Thank you for your attention, do you have any questions ?

References


Mathis Bouverot-Dupuis and Yannick Forster.
Code Generation via Meta-programming in Dependently Typed Proof Assistants.


Emilio Jesús Gallego Arias, Ali Caglayan, Shachar Itzhaky, Fréderic Blanqui, Rodolphe Lepigre, et al.
rocq-lsp: a Language Server for the Rocq Prover, 2025.


Titouan Lozac'h and Nicolas Magaud.
Post-processing Coq Proof Scripts to Make Them More Robust.
In *2nd Workshop on the development, maintenance, refactoring and search of large libraries of proofs , September 13-14, 2024, Tbilissi, Georgia*, 2024.


Nicolas Magaud.
Towards Automatic Transformations of Coq Proof Scripts.
In *Automated Deduction in Geometry (ADG 2023)*, Belgrade, Serbia, November 2023. Pedro Quaresma et Kovács Zoltán.


Jessica Shi, Cassia Torczon, Harrison Goldstein, Andrew Head, and Benjamin Pierce.
Designing Proof Deautomation in Rocq.
In *Proceedings of the 15th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU)*, 2025.