The Coq Proof Script Visualiser (coq-psv)

Coq Workshop 2020, Virtual

Mario Frank
mario.frank@uni-potsdam.de

Potsdam University
Institute for Computer Science

July 5, 2020

Inhaltsverzeichnis

Coq is powerful

Coq enables users to

- formalise properties of software/hardware/...
- interactively prove those properties
- exchange formalisations and proofs (quite easily)
- generate printable variants of proof scripts (coqdoc)
- even do all this online (JSCoq)

Coq is powerful ... but

Coq enables users to

- formalise properties of software/hardware/...
- interactively prove those properties
- exchange formalisations and proofs (quite? easily)
- generate (restricted) printable variants of proof scripts (coqdoc)
- even do all this (for one file) online (JSCoq)

but the coqdoc output contains only the used tactics, i.e. goals/hyps per step only in live session
$\rightarrow$ the recipient needs (to install/use) a compatible version of Coq
$\rightarrow$ not suitable for "offline" use (as pdf)

Live sessions are great ... but

There are some problems (with exchange of vernacular files)

- find compatible Coq version
- installation necessary (may be problematic/frustrating for some OS)
- or use JSCoq, if compatible

but sometimes you do not want live sessions

- when including parts of a proof in a paper/thesis
- using proof scripts in offline teaching (e.g. as cloze)

Proof script excerpts ... why?

A reader/reviewer of a paper, thesis, ..., may want to have a selfcontaining document concerning the presentation of

- the functionality of (new) tactics
- the main structure of a proof
- relevant details about a proof

But neither coqdoc, nor other tools (e.g. Proviola) do generate an output including all goals and hyps for each step (for offline use)
$\rightarrow$ Typesetting for "offline" use may be cumbersome as you have to do it by hand.

Proof scripts as Cloze? .. Yeah, we did that

1. Give students a partially filled proof and let them fill the gaps
2. (hopefully) improves understanding of the process of proving

$+_{1/2}$	$\forall\ m\ l : \mathbb{N}, 0 \oplus m = 0 \oplus l \rightarrow m = l$
prove_all_imp_star.	**A** **B** $m = l$
drop_identities in H.	**A** $H \ : m = l$ $m = l$

Figure: a proof cloze

Further uses: Provide the enriched proof script and let students write an equivalent textbook proof (We did that, too.)

Aims

Extract the information about a proof including

- the used tactic
- the resulting hypotheses
- the resulting goals

for each step and

- represent it as LaTeX table
- with (almost) no interaction by the user
- without the need to do any manipulation on the output

We (partially) succeeded

Extract the information about a proof including

- the used tactic $\surd$
- the resulting hypotheses $\surd$
- the resulting goals $\surd$

for each step and

- represent it as LaTeX table $\surd$
- with (almost) no interaction by the user $\surd$
- without the need to do grave manipulation on the output $(\surd)$

General Concept

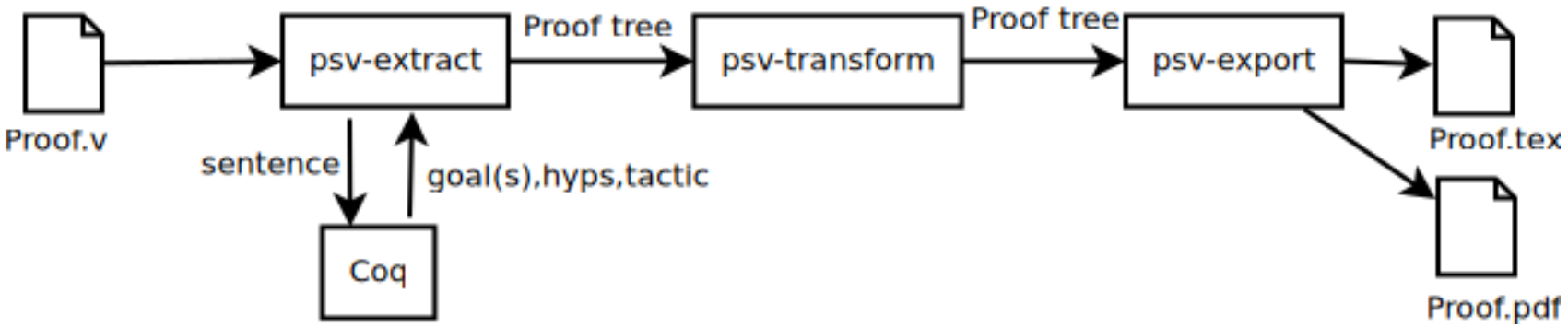

Figure: The general workflow (for one file)

Works analogously for complete projects

Extraction

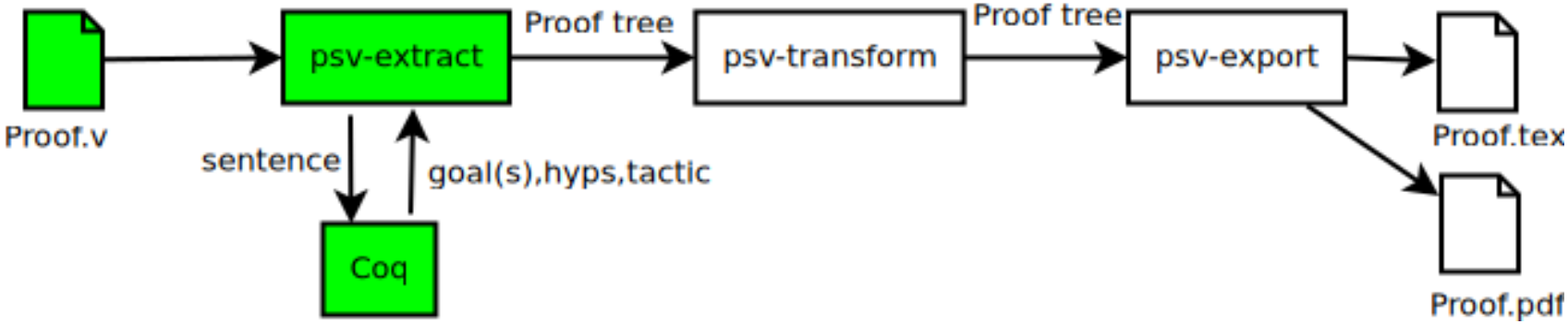

Figure: The general workflow (extraction)

Extraction

Given an (independent) Vernacular file

1. feed the file sentence-wise into the Coq parsing routine
2. if a theorem statement is given, switch into proof mode
3. store the statement information (statement, name)
4. process the "Proof." command (or equivalent) and gather the initial goal and hypotheses (as proof tree node)
5. for each following step (until QED/Admitted) do the same
6. when QED/Admitted is recognised, leave the proof mode (and store this info)
7. seek the next theorem (and finally find the end of the file)
8. handle the proof tree(s) to pqv-transform

Demo - General output

DEMO

Transformation

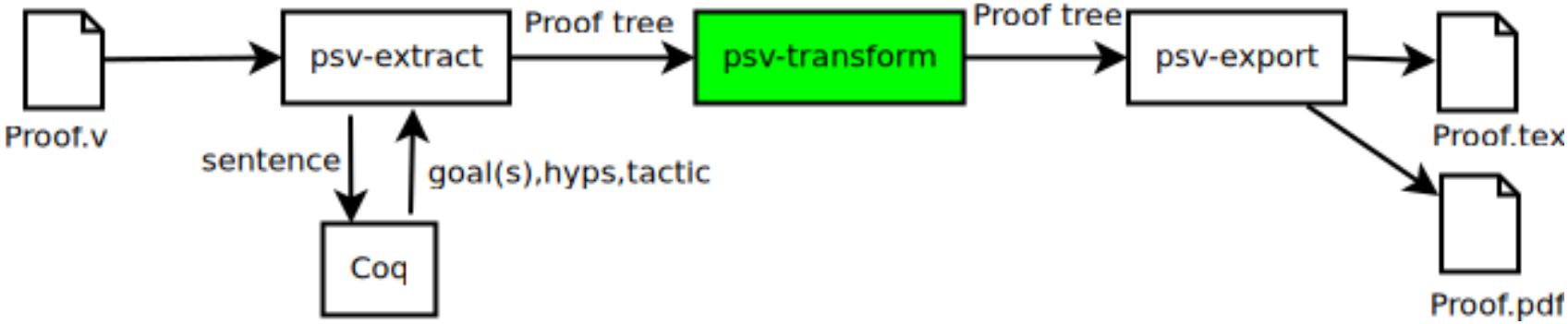

Figure: The general workflow (transformation)

Transformation : General

Some information is superfluous or makes the output less readable

1. goals after using some induction tactic
   if handled by bullets later on
2. invariant hypotheses
   those that do not change after introduction
3. singleton clear/rename/move sentences (currently unsolved)

Using command line options,

1. can be hidden
2. can be boxed on introduction and hidden afterwards

Example - Transformation

Example

Export

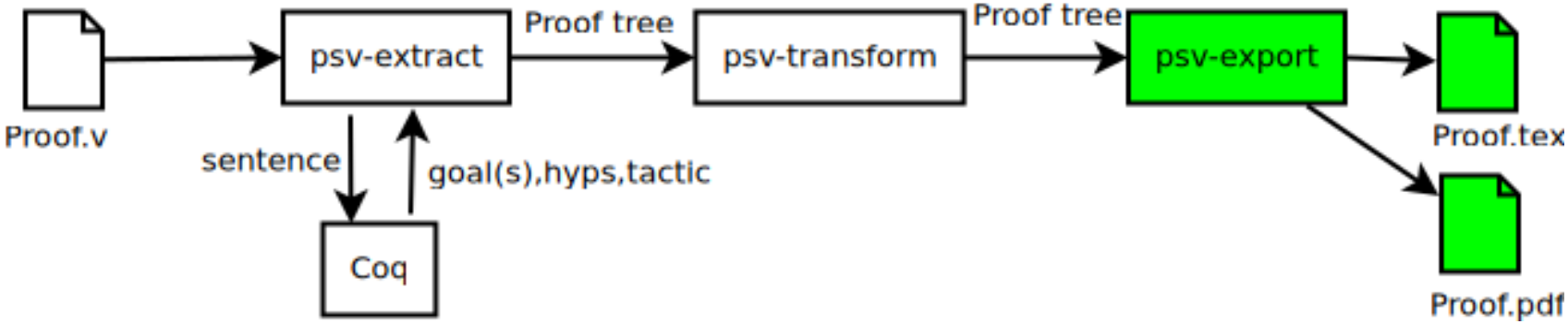

Figure: The general workflow (export)

Export : General

For LaTeX output, we use longtables (multipage tables)
It is possible, to generate one standalone/includable

1. file containing all proof tables
2. file for each proof in the original file

and additionally the respective PDF files (via pdflatex)
We support two output flavours

1. the Coq style (as seen)
2. the sequent style (more condensed)

Example - Sequent Style output

Example

Generating LaTeX

Coq-psv provides 3 LaTeX-template files

1. a table template file (is filled with information from the proof tree)
2. a command template file (with default commands for spacings, can be adopted by the user)
3. a standalone document template, is filled with the latter one

Supported Platforms

- Coq-psv is implemented in OCaml and works with Coq 8.10.
- Coq 8.11 is currently not supported (lack of time, upgrade path unclear)
- Installation from opam repository

Current State

- LaTeX and PDF support for single files and complete projects
- many customisations on template files or by command line options
- Output is quite readable for some proof styles (e.g. using medium degree tactics[Böh19])

Current Problems

- Eliminating clear/rename/move is not possible - it is not clear how to detect the type of tactic and its arguments
- Handling Focus commands is not optimal
- vertical alignment of the tactic column is not always centered $\rightarrow$ Does not look so nice for proofs like in mathcomp
- If processing multiple files, the dependencies to pevious files are not resolved (help needed)

Current/Future Work

- In progress: Refactoring of proofs (introducing bullets,...)
- HTML support
- Extracting only designated proofs
- Integration into Coq and CoqIDE

Special Thanks

- Sebastian Böhne - For providing the idea and some requirements
- Chris Dams - For providing a toy example file (formalisation of the Nim game)
- Emilio Jesús Gallego Arias - For giving useful hints and offering to extend SerAPI

Sebastian Böhne.
Different Degrees of Formality – An Introduction to the Concept and a Demonstration of its Usefulness.
Phd thesis, Universität Potsdam, 2019.

Problem : Coq is versatile

Normally, this is great ... but:
If there are multiple goals, you can handle them:

1. individually by bullets (great!)
2. individually by subproofs/brackets (also fine!)
3. individually by (deprecated) Focus commands
4. not at all individually (a tactic always addresses the first goal)

Even worse: A "Focus"ed goal can be unfocused without completion
$\rightarrow$ How to build a proof tree?

(Partial) Solution

The easy catch:

- Using bullets and subproofs is fine - creating a tree structure is easy (end of subproof is signalled by Coq).
- No individual goal handling is also okay $\rightarrow$ sequence of proof nodes

Focus is more problematic:

1. the goal can be unfocused without completion (and continued later)
2. no signal about completion by Coq
3. deprecated since version 8.9

$\rightarrow$ we handle this case as sequence of proof tree nodes.

What is a proof tree node, anyway?

Proof Tree nodes

```
type ptree_node =
| Leaf of      (psit * bool)
| Sequence of (psit * ptree_node)
| Branch of    (psit * ptree_node list)
| Split of     (psit * (ptree_node * ptree_node))
```

Figure: Definition of a proof tree node

1. psit : tactic, (list of) goals and hyps
2. Branch : the bullet case
3. Split : the subproof case ({... * }...)
4. Leaf : The end of a subproof handled by bullet/bracket (and if it closes the overall proof)
5. Sequence : unstructured part

Detecting/Marking invariant hypotheses

Recap: A hypothesis is invariant, if it is not modified after creation.
Given a proof tree,

1. traverse the tree
2. for each tree node, get all newly introduced hypotheses (done by a modified version of the Coq diff algorithm)
3. mark the hypotheses with a reference to a property "invariant" (default true)
4. store a list of tuples (hyp, ref invariant)
5. if one of the hypotheses is removed/changed, toggle the invariance property of it.

- Not purely functional
+ quite fast

Transformation : Output Example

prove_by_induction.	Hidden 2 goal(s)
$+_{1/2}$	$\forall\ K : List\ \mathbb{N},\ list\_in\_strorder\ [] \to list\_in\_strorder\ K \to [] \subseteq K \to K \subseteq [] \to [] = \{List\ \mathbb{N}\}\ K$
prove_all_imp_star.	K : $List\ \mathbb{N}$ H : $list\_in\_strorder\ []$ H0 : $list\_in\_strorder\ K$ H1 : $[] \subseteq K$ $H2 : K \subseteq []$ --- $[] = \{List\ \mathbb{N}\}\ K$
exception H2.	$H2 : K = \{List\ \mathbb{N}\}\ []$ --- $[] = \{List\ \mathbb{N}\}\ K$

Figure: Hiding invariants and superfluous proof situations

Export: Sequent style example

Next step in Coq	Proof situation
Proof.	$\vdash \forall\ L\ K : List\ \mathbb{N},\ list\_in\_strorder\ L \to list\_in\_strorder\ K \to L \subseteq K \to K \subseteq L \to L = \{List\ \mathbb{N}\}\ K$
prove_by_induction.	Hidden 2 goal(s)
$+_{1/2}$	$\vdash \forall\ K : List\ \mathbb{N},\ list\_in\_strorder\ [] \to list\_in\_strorder\ K \to [] \subseteq K \to K \subseteq [] \to [] = \{List\ \mathbb{N}\}\ K$
prove_all_imp_star.	$\boxed{\text{K}} : List\ \mathbb{N}\ ;\ \boxed{\text{H}} : list\_in\_strorder\ []\ ;\ \boxed{\text{H0}} : list\_in\_strorder\ K\ ;\ \boxed{\text{H1}} : [] \subseteq K\ ;\ H2 : K \subseteq [] \vdash [] = \{List\ \mathbb{N}\}\ K$
exception H2.	$H2 : K = \{List\ \mathbb{N}\}\ [] \vdash [] = \{List\ \mathbb{N}\}\ K$

Figure: Hiding invariants and superfluous proof situations (sequent style)