Improvement of the automation in the `coq-waterproof` library

Balthazar Patiachvili
Under the supervision of Jim Portegies

ENS Paris-Saclay & TU/e

September 5, 2023

Contents

1 Introduction

2 Automation control

3 Automated rewriting

4 Optimization

5 Conclusion

Context (1)

Issue

Learning how to perform logically coherent reasoning

Context (1)

Issue

Learning how to perform logically coherent reasoning

Learning process

- Can be challenging for undergraduate science students

Context (1)

Issue

Learning how to perform logically coherent reasoning

Learning process

- Can be challenging for undergraduate science students
- Solution: use proof assistants as a pedagogical tool, as Coq in [Kno+17] or Lean in [TI21]

Context (2)

Benefits of proof assistants

- Ensure the validity of every step of the proof
- Real-time feedback on user's actions

Context (2)

Benefits of proof assistants

- Ensure the validity of every step of the proof
- Real-time feedback on user's actions

Downsides of proof assistants

- Confusing syntax for inexperienced users
- Do not guarantee to improve handwritten proofs [Kno+17]

Waterproof

Presentation

- Educational software created by members of the TU/e [Wem+22], in particular Jim Portegies and Jelle Wemmenhove

Waterproof

Presentation

- Educational software created by members of the TU/e [Wem+22], in particular Jim Portegies and Jelle Wemmenhove
- Proof assistant in natural language based on `coq-waterproof` (Coq library written in Ltac2)

Waterproof

Presentation

- Educational software created by members of the TU/e [Wem+22], in particular Jim Portegies and Jelle Wemmenhove
- Proof assistant in natural language based on `coq-waterproof` (Coq library written in Ltac2)
- Has already been used for some years as an option for a analysis course

Waterproof

Presentation

- Educational software created by members of the TU/e [Wem+22], in particular Jim Portegies and Jelle Wemmenhove
- Proof assistant in natural language based on `coq-waterproof` (Coq library written in Ltac2)
- Has already been used for some years as an option for a analysis course
- Focus on the accessibility for non-expert users and on the resemblance to handwritten proofs

Example of a proof in Coq and in Waterproof

Coq proof of $\forall n, m \in \mathbb{N}, n = 0 \Longrightarrow m + 1 \neq n$

```
Goal forall n m: nat, n = 0 -> S m <> n.
Proof.
  intros n m H H'.
  rewrite H in H'.
  inversion H'.
Qed.
```

Waterproof proof of $\forall n, m \in \mathbb{N}, n = 0 \Longrightarrow m + 1 \neq n$

```
Goal forall n m: nat, n = 0 -> S m <> n.
Proof.
  Take n, m: nat.
  Assume that (n = 0) (i).
  By (i) we conclude that (S m <> n).
Qed.
```

Internship

Proofs automation

- *waterprove*: tactic used to solve automatically goals

Internship

Proofs automation

- *waterprove*: tactic used to solve automatically goals
- Need to have automatization to skip non-interesting parts of the proof (e.g $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0 \implies \frac{\varepsilon}{2} > 0$)

Internship

Proofs automation

- *waterprove*: tactic used to solve automatically goals
- Need to have automatization to skip non-interesting parts of the proof (e.g $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0 \implies \frac{\varepsilon}{2} > 0$)
- Need to be able to control the automatization

Internship

Proofs automation

- *waterprove*: tactic used to solve automatically goals
- Need to have automatization to skip non-interesting parts of the proof (e.g $\forall \varepsilon \in \mathbb{R}, \varepsilon > 0 \implies \frac{\varepsilon}{2} > 0$)
- Need to be able to control the automatization

$\implies$ Two main axes of improvement: control and reinforcement of the automation

Contents

1 Introduction

2 Automation control

3 Automated rewriting

4 Optimization

5 Conclusion

Automation control

Idea

- Automatic proofs are done by "searching" a proof in the same way as prolog

Automation control

Idea

- Automatic proofs are done by "searching" a proof in the same way as prolog
- Have more control on proof search flow

Automation control

Idea

- Automatic proofs are done by "searching" a proof in the same way as prolog
- Have more control on proof search flow
  $\longrightarrow$ Skip some parts of the search, reject proofs without a certain property, ...

Automation control

Idea

- Automatic proofs are done by "searching" a proof in the same way as prolog
- Have more control on proof search flow
  $\longrightarrow$ Skip some parts of the search, reject proofs without a certain property, ...

Proof of concept

Reject proofs where the user gives a lemma that is not used

Automation control

Idea

- Automatic proofs are done by "searching" a proof in the same way as prolog
- Have more control on proof search flow
  $\longrightarrow$ Skip some parts of the search, reject proofs without a certain property, ...

Proof of concept

Reject proofs where the user gives a lemma that is not used

Example of a proof that should be rejected

```
Goal sin 0 = 0.
Proof.
  auto using cos_0, sin_0.
Qed.
```

Prolog (1)

Description

Logic programming language based on first-order logic used to solve problems involving objects and relationships

Prolog (1)

Description

Logic programming language based on first-order logic used to solve problems involving objects and relationships

Example of a prolog program

```
mother_child(alice, david). % (1)
father_child(charlie, david). % (2)
mother_child(alice, bob). % (3)

parent_child(X, Y) :- father_child(X, Y). % (4)
parent_child(X, Y) :- mother_child(X, Y). % (5)
child_parent(X, Y) :- parent_child(Y, X). % (6)
```

Prolog (2)

Example of a prolog query

```
mother_child(alice, david). % (1)
father_child(charlie, david). % (2)
mother_child(alice, bob). % (3)

parent_child(X, Y) :- father_child(X, Y). % (4)
parent_child(X, Y) :- mother_child(X, Y). % (5)
child_parent(X, Y) :- parent_child(Y, X). % (6)

?- child_parent(bob, alice).
```

Prolog (2)

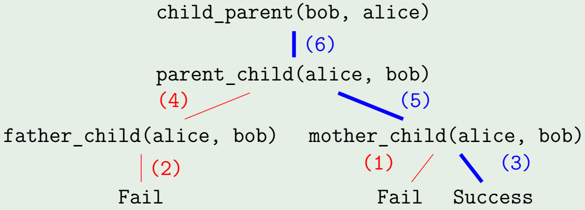
Example of a prolog query

```
mother_child(alice, david). % (1)
father_child(charlie, david). % (2)
mother_child(alice, bob). % (3)

parent_child(X, Y) :- father_child(X, Y). % (4)
parent_child(X, Y) :- mother_child(X, Y). % (5)
child_parent(X, Y) :- parent_child(Y, X). % (6)

?- child_parent(bob, alice).
```

Proof search tree of the query



auto

auto tactic

- Works on the same principle

auto

auto tactic

- Works on the same principle
- *rule* (Prolog) $\longrightarrow$ *hint* (auto)

auto

auto tactic

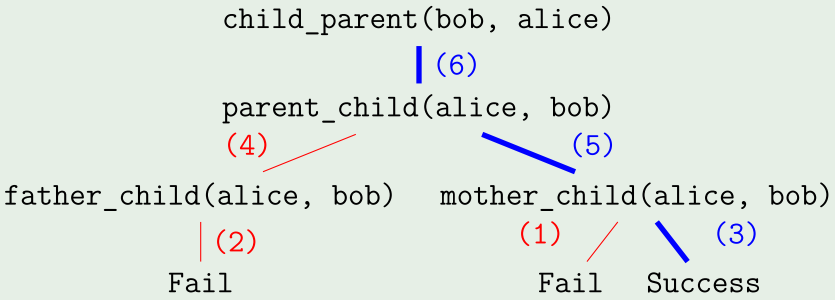
- Works on the same principle
- *rule* (Prolog) $\longrightarrow$ *hint* (auto)

Trace

Ordered list of tuples containing (at least) the tried hints who leads or whose parent leads to a complete proof, and booleans indicating for each hint if it is used for the final proof or not

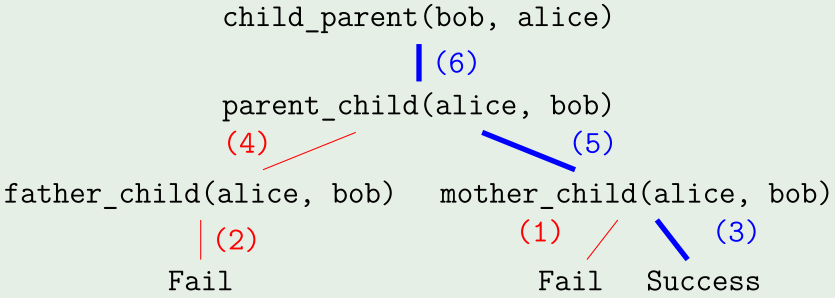
Trace

Trace of a proof search tree



Trace

Trace of a proof search tree



Trace:

`[(6, true); (4, false); (5, true); (1, false); (3, true)]`

Control at the end of the proof (contribution)

Idea

Make the proof search fail if a given lemma has not been used

Control at the end of the proof (contribution)

Idea

Make the proof search fail if a given lemma has not been used

- Retrieve the full trace of the proof search

Control at the end of the proof (contribution)

Idea

Make the proof search fail if a given lemma has not been used

- Retrieve the full trace of the proof search
- After the proof search, check if each given lemma has been used

Control at the end of the proof (contribution)

Idea

Make the proof search fail if a given lemma has not been used

- Retrieve the full trace of the proof search
- After the proof search, check if each given lemma has been used

Example of a proof rejection because of an unused lemma

```
Goal forall n: nat, n = n.
Proof.
    Take n: nat.
    Fail By f_equal we conclude that (n = n).
    We conclude that (n = n).
Qed.
```

Trace: `[(assumption, false); (intro; false); (@eq_refl, true)]`

Control during the proof search (contribution) (1)

Idea

- A satisfying proof is not always the first found

Control during the proof search (contribution) (1)

Idea

- A satisfying proof is not always the first found
- Keep the previous idea of the control of the proof, but making the checks during the proof search

Control during the proof search (contribution) (1)

Idea

- A satisfying proof is not always the first found
- Keep the previous idea of the control of the proof, but making the checks during the proof search
- Continue the proof search in case of failure

Control during the proof search (contribution) (1)

Idea

- A satisfying proof is not always the first found
- Keep the previous idea of the control of the proof, but making the checks during the proof search
- Continue the proof search in case of failure
- Need to transmit informations through proof search flow
  $\longrightarrow$ Typed tactics (generalization of the OCaml tactic monad)

Control during the proof search (contribution) (2)

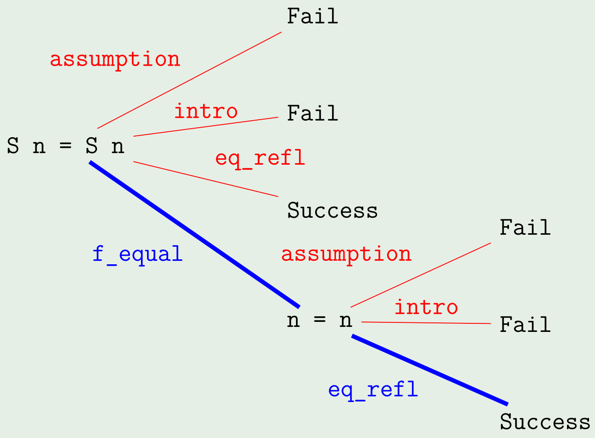
Example of the goal `forall n: nat, S n = S n`

```
Goal forall n: nat, S n = S n.
Proof.
  intros n.
  apply eq_refl.
Qed.
```

```
Goal forall n: nat, S n = S n.
Proof.
  intros n.
  apply f_equal.
  apply eq_refl.
Qed.
```

Control during the proof search (contribution) (2)

Example of the goal `forall n: nat, S n = S n`



Control during the proof search (contribution) (3)

Possible improvement

- Some parts of the proof search tree are currently skipped

Control during the proof search (contribution) (3)

Possible improvement

- Some parts of the proof search tree are currently skipped
- In practice, this edge case never happened in our cases

Control during the proof search (contribution) (3)

Possible improvement

- Some parts of the proof search tree are currently skipped
- In practice, this edge case never happened in our cases
- Would need a complete rewrite of our implementation of `auto`

Contents

1. Introduction
2. Automation control
3. Automated rewriting
4. Optimization
5. Conclusion

Automated rewriting

Idea

- Improve proof search strength $\longrightarrow$ solve more goals automatically

Automated rewriting

Idea

- Improve proof search strength $\longrightarrow$ solve more goals automatically
- Use rewrites during the proof search

Automated rewriting

Idea

- Improve proof search strength $\longrightarrow$ solve more goals automatically
- Use rewrites during the proof search
- Automatically generate rewrite hints for `autorewrite` .

Automated rewriting

Idea

- Improve proof search strength ⟶ solve more goals automatically
- Use rewrites during the proof search
- Automatically generate rewrite hints for `autorewrite` .

Example of a goal that cannot be solve automatically currently

```
Goal forall x: R, x = 0 -> sin x = 0.
Proof.
  intros x H.
  Fail progress (auto using sin_0).
  rewrite H; auto using sin_0.
Qed.
```

Rewriting

`rewrite`

Replace subterms in a given expression with other subterms that have be proven to be equal [Coqa]

Rewriting

`rewrite`

Replace subterms in a given expression with other subterms that have be proven to be equal [Coqa]

Example of a use of `rewrite`

```
x, y, z:  R
f:  R -> R
H: x = y
---------------
f x = f z
```

`rewrite H.` ⟶

```
x, y, z:  R
f:  R -> R
H: x = y
---------------
f y = f z
```

Rewriting

`rewrite`

Replace subterms in a given expression with other subterms that have be proven to be equal [Coqa]

Example of a use of `rewrite`

```
x, y, z:  R                              x, y, z:  R
f:  R -> R       rewrite H.              f:  R -> R
H: x = y      ---------------->          H: x = y
-----------                              -----------
f x = f z                                f y = f z
```

`autorewrite`

- Apply rewritings based on the given rewrite hints

Rewriting

`rewrite`

Replace subterms in a given expression with other subterms that have be proven to be equal [Coqa]

Example of a use of `rewrite`

```
x, y, z:  R
f:  R -> R
H: x = y
____________
f x = f z
```

`rewrite H.` ⟶

```
x, y, z:  R
f:  R -> R
H: x = y
____________
f y = f z
```

`autorewrite`

- Apply rewritings based on the given rewrite hints
- Can apply another given tactic between each rewrite

Rewriting

`rewrite`

Replace subterms in a given expression with other subterms that have be proven to be equal [Coqa]

Example of a use of `rewrite`

```
x, y, z:  R                        x, y, z:  R
f:  R -> R       rewrite H.        f:  R -> R
H: x = y      ---------------->    H: x = y
-----------                        -----------
f x = f z                          f y = f z
```

`autorewrite`

- Apply rewritings based on the given rewrite hints
- Can apply another given tactic between each rewrite
  ⟶ use with our version of `auto`

Automated use of hypotheses (contribution) (1)

Idea

- `autorewrite` is useful but rewrite hints must be declared before its use

Automated use of hypotheses (contribution) (1)

Idea

- `autorewrite` is useful but rewrite hints must be declared before its use
- Do the same as `auto` : use current hypotheses

Automated use of hypotheses (contribution) (1)

Idea

- `autorewrite` is useful but rewrite hints must be declared before its use
- Do the same as `auto` : use current hypotheses

waterprove

Call to our own version of `autorewrite` calling as argument our version of `auto`

Automated use of hypotheses (contribution) (2)

Example of a proof where auto fails but waterprove succeeds

```
Goal forall A: Set, forall x y z: A, forall f: A -> A,
  x = y -> f y = f z -> f x = f z.
Proof.
  intros A x y z f H1 H2.
  Fail progress auto.
  waterprove.
Qed.
```

Automated use of hypotheses (contribution) (2)

Example of a proof where auto fails but waterprove succeeds

```
Goal forall A: Set, forall x y z: A, forall f: A -> A,
  x = y -> f y = f z -> f x = f z.
Proof.
  intros A x y z f H1 H2.
  Fail progress auto.
  waterprove.
Qed.
```

Possible improvement

Extend the work done on automation control to our version of `autorewrite`

Contents

1 Introduction

2 Automation control

3 Automated rewriting

4 Optimization

5 Conclusion

Branch skipping (contribution) (1)

Issue

- Compilation time undetermined ($>$ 15 minutes)
- Very high amount of hints tried ($>$ 10,000,000 against 2,000,000 usually)

Branch skipping (contribution) (1)

Issue

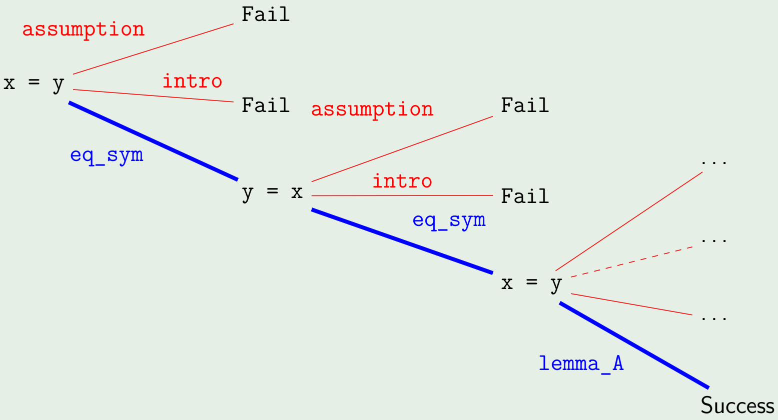
- Compilation time undetermined ($>$ 15 minutes)
- Very high amount of hints tried ($>$ 10,000,000 against 2,000,000 usually)

Idea

Skip branches in the proof search tree leading to proof states already visited

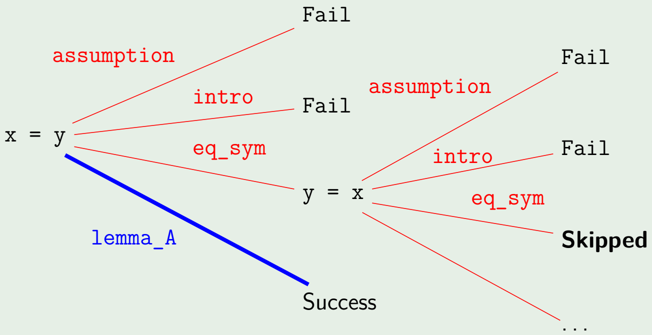
Branch skipping (contribution) (2)

Example of a proof search tree with and without the optimization



Branch skipping (contribution) (2)

Example of a proof search tree with and without the optimization



Branch skipping (contribution) (2)

Conclusion

- The issue was caused by a bug found and fixed later.

Branch skipping (contribution) (2)

Conclusion

- The issue was caused by a bug found and fixed later.
- Still improvements are visible : $\sim 1,260,000$ hints tried without against $\sim 670,000$ hints tried with optimization

Branch skipping (contribution) (2)

Conclusion

- The issue was caused by a bug found and fixed later.
- Still improvements are visible : $\sim 1,260,000$ hints tried without against $\sim 670,000$ hints tried with optimization
- Without the file `tests/tactics/ItHolds.v`: $\sim 208,000$ without against $\sim 154,000$ with the optimization

Contents

1. Introduction
2. Automation control
3. Automated rewriting
4. Optimization
5. Conclusion

Conclusion

- Several improvements have been made to the automation system: control of the proof search flow and add of automatic rewritings during the proof searches

Conclusion

- Several improvements have been made to the automation system: control of the proof search flow and add of automatic rewritings during the proof searches
- Generalization of the existing OCaml tactic monad

Conclusion

- Several improvements have been made to the automation system: control of the proof search flow and add of automatic rewritings during the proof searches
- Generalization of the existing OCaml tactic monad
- Optimization of the proof searches with a notable reduction of tried hints

Conclusion

- Several improvements have been made to the automation system: control of the proof search flow and add of automatic rewritings during the proof searches
- Generalization of the existing OCaml tactic monad
- Optimization of the proof searches with a notable reduction of tried hints

- Some fixes have to be done to complete the work done

Conclusion

- Several improvements have been made to the automation system: control of the proof search flow and add of automatic rewritings during the proof searches
- Generalization of the existing OCaml tactic monad
- Optimization of the proof searches with a notable reduction of tried hints

- Some fixes have to be done to complete the work done

- Further research and development: use the tools made during this internship to improve the practicality for both students and teachers

Conclusion

- Several improvements have been made to the automation system: control of the proof search flow and add of automatic rewritings during the proof searches
- Generalization of the existing OCaml tactic monad
- Optimization of the proof searches with a notable reduction of tried hints

- Some fixes have to be done to complete the work done

- Further research and development: use the tools made during this internship to improve the practicality for both students and teachers
- `coq-waterproof` has been added to opam's repository

Thanks for your attention

[CM84] William F. Clocksin and Christopher S. Mellish. *Programming in Prolog*. Springer Berlin Heidelberg, 1984. DOI: 10.1007/978-3-642-96661-3. URL: https://doi.org/10.1007/978-3-642-96661-3.

[Coqa] Coq Team. *Coq's Reference Manual*. URL: https://coq.inria.fr/distrib/current/refman/.

[Coqb] Coq Team. *Coq's source code*. URL: https://github.com/coq/coq.

[Kai+18] Jan-Oliver Kaiser et al. "Mtac2: Typed Tactics for Backward Reasoning in Coq". In: *Proc. ACM Program. Lang.* 2.ICFP (July 2018). DOI: 10.1145/3236773. URL: https://doi.org/10.1145/3236773.

[Kno+17] Maria Knobelsdorf et al. "Theorem Provers as a Learning Tool in Theory of Computation". In: *Proceedings of the 2017 ACM Conference on International Computing Education Research*. ICER '17. Tacoma, Washington, USA: Association for Computing Machinery, 2017, pp. 83–92. ISBN: 9781450349680. DOI: 10.1145/3105726.3106184. URL: https://doi.org/10.1145/3105726.3106184.

[TI21] Athina Thoma and Paola Iannone. “Learning about Proof with the Theorem Prover LEAN: the Abundant Numbers Task”. In: *International Journal of Research in Undergraduate Mathematics Education* 8.1 (July 2021), pp. 64–93. DOI: 10.1007/s40753-021-00140-1. URL: https://doi.org/10.1007/s40753-021-00140-1.

[Wem+22] Jelle Wemmenhove et al. *Waterproof: educational software for learning how to write mathematical proofs*. 2022. arXiv: 2211.13513 [math.HO].

Typed tactic functor

```
module type Mergeable = sig
    type elt
    val empty : elt
    val merge : elt -> elt -> elt
end

(** Generalization of tactics defined in coq-core for {! Mergeable}-typed tactics *)
module TypedTactics(M: Mergeable) = struct

  (** Merge of tactics' returned elements *)
  let typedThen (tactic1: M.elt tactic) (tactic2: M.elt tactic): M.elt tactic =
    tactic1 >>= fun elt1 ->
    tactic2 >>= fun elt2 ->
    tclUNIT @@ M.merge elt1 elt2

  (** Same as {! typedThen} with a list of tactics *)
  let typedLongThen (tactics: M.elt tactic list): M.elt tactic =
    List.fold_left typedThen (tclUNIT M.empty) tactics

  (** Generalization of {! Proofview.Goal.enter} *)
  let typedGoalEnter (f: Goal.t -> M.elt tactic): M.elt tactic =
    Goal.goals >>= fun goals ->
    let tactics = List.map (fun goal_tactic -> goal_tactic >>= f) goals in
    List.fold_left (fun acc tac -> typedThen acc tac) (tclUNIT M.empty) tactics

  (** Generalization of {! Proofview.tclINDEPENDENT} *)
  let typedIndependent (tactic: M.elt tactic): M.elt tactic =
    tclINDEPENDENTL tactic >>= fun elts -> tclUNIT @@ List.fold_left M.merge M.empty elts

end
```

Control failure