

Making proofs in Coq

Yves Bertot

Goal directed proof

- ▶ In theory, proving is the same as programming
- ▶ In practice, intermediate statements are more relevant than proof constructs
- ▶ Procedural approach
 1. State an initial statement
 2. Apply a command that decomposes a statement into easier ones
 3. repeat step 2
- ▶ Sometimes step 2 does not produce new statements
- ▶ When no more subgoals, the proof must be saved using Qed.
- ▶ Proof scripts record only the commands that have been applied
- ▶ Difficult reading, script management is needed

Start a proof

- ▶ Lemma *name* : *formula*.

=====

formula

- ▶ The name must be new
- ▶ The formula must be well-formed
- ▶ Other keywords can be used
 - ▶ Theorem, Fact, Example

Decomposing a logical formula

- ▶ Example: $A \wedge B$
- ▶ We want to prove A and B as one formula
- ▶ But logically, it is enough to prove A and B separately
- ▶ To go from $A \wedge B$ to A and B requires a logical step
- ▶ This example was about a *conclusion*, we can have similar problems when $A \wedge B$ appears as an hypothesis

Hypotheses and conclusion

- ▶ During a proof, Coq displays *goals*
- ▶ Each goal contains a conclusion: the formula to prove
- ▶ Each goal also contains a *context* made of *hypotheses*
 - ▶ Each hypothesis has a name and a statement
- ▶ Example

H1 : $x \leq y$

H2 : $y \leq z$

=====

$x \leq z$

Using the context

- ▶ Hypotheses are meant to be used to prove the current goal
- ▶ When an hypothesis H matches the goal exactly, use exact H.
- ▶ You can also use assumption.

▶ H : A

=====

A

exact H.

the goal is solved!

- ▶ Exact matching may involve computation

▶ H : P 3

=====

P (2 + 1)

assumption.

the goal is solved!

Tactics for universal quantification (in conclusion)

- ▶ How do we prove $\text{forall } x:T, A \ x$?
 - ▶ Reason on an arbitrary member of type T
 - ▶ Arbitrary: we don't know anything about it, its *new*
- ▶ Tactic : `intros`
- ▶

```
=====
forall x : T, A x
intros y.
y : T
=====
A y
```
- ▶ y must not be in the context (it must be *fresh*)
- ▶ usually, we use directly the name x

Implication (in conclusion)

- ▶ How do we prove that $A \rightarrow B$ holds?
 - ▶ We assume we know A , and then we look at just B
- ▶ Add A to the known facts (the context)
- ▶ intro H (the name H must be fresh)

Universal quantification (in hypotheses)

► How to use `forall x : T, A x -> B x`?

► In particular if we have to prove `B e`

► `H : forall x : T, A x -> B x`

=====

`B e`

`apply H.`

`H : forall x : T, A x -> B x`

=====

`A e`

► Coq guesses that `H` is used on `e`

► Beware! `apply` handles all universal quantifications and implications in one round

- Guess values of universally quantified variables
- Create a new goal for every premise of an implication

Missing universally quantified variables

- ▶ The guess work is done by matching the theorem's conclusion with the goal's conclusion
- ▶ Hopefully, all universally quantified variable can be determined
- ▶ missing variables can be given by the user
- ▶ Example

```
Require Import ZArith.  Open Scope Z_scope.  
Check Zle_trans.  
Zle_trans :  
  forall x y z : Z, x <= y -> y <= z -> x <= z.
```

- ▶ This theorem can be used in `apply` (like any hypothesis)
- ▶ The variable `y` does not occur in the theorem's conclusion.

Giving missing variables

- ▶ `Zle_trans` :
 `forall x y z : Z, x <= y -> y <= z -> x <= z.`
- ▶ First syntax: by name
 `apply Zle_trans with (y:= formula)`
- ▶ Second syntax: by hypothesis
 `H : x <= 3`
 `=====`
 `x <= 10`
 `apply Zle_trans with (1:=H).`
 `H : x <= 3`
 `=====`
 `3 <= 10`
- ▶ Third syntax: by application
 `apply (Zle_trans x 3) or apply (Zle_trans _ 3)`
- ▶ Universally quantified theorems can be used like functions!

Implications (in hypotheses)

- ▶ A particular case of apply
- ▶ No variable needs guessing
- ▶ has many new goals as there are premises
- ▶ A particular case: when no implication (no premise), apply works, but exact is more explicit

using implications and quantifications without the conclusion

- ▶ Add explicitly consequences using assert

- ▶ $H : A \rightarrow B$

$Ha : A$

=====

C

assert ($H' : B$).

=====

B

apply H.

=====

A

- ▶ A second goal has an hypothesis H' stating B
- ▶ Implication and quantification theorems may be used as functions

assert ($H' := H Ha$).

Conjunction

- ▶ Prove $A \wedge B$
`split`
- ▶ Use $H : A \wedge B$
`destruct H as [H1 H2]` or `case H`
 - ▶ creates two hypotheses $H1 : A$ and $H2 : B$
 - ▶ the names $H1$ and $H2$ have to be fresh
- ▶ Behavior intuitive: replace connectives by their meaning
- ▶ Name of tactics needs to be remembered...

disjunction

- ▶ Prove $A \vee B$
- ▶ Choose to prove A or to prove B
`left` or `right`
- ▶ Use $H : A \vee B$
`destruct H as [H1 | H2]` or `case H`
 - ▶ Two goals generated, one where A is
 - ▶ Need to cover all possibilities
- ▶ Same tactic names as for conjunction

Short cut for destruct

- ▶ In presence of nested logical connectives
- ▶ frequent situation `destruct H as [H1 H2]` followed by `destruct H1 as [H3 | H4]`
- ▶ Abbreviated as `destruct H as [[H3 | H4] H2]`
 - ▶ Two goals, one with H3 and H2, the other with H4 and H2
- ▶ Second frequent situation `intros H` followed by `destruct H as [H1 H2]`
- ▶ abbreviated as `intros [H1 H2]`.

Combining tactics

- ▶ Use several tactics in one command
- ▶ `tac1; tac2,`
`tac2` is used on all goals generated by `tac1`
- ▶ `tac; [tac1 | ... | tacn],`
`taci` is applied on the i^{th} generated goal

demonstration

```
Lemma example : forall A B P Q, (A  $\vee$  B)  $\wedge$ 
  (forall x:nat, P x  $\vee$  Q x) ->
  forall x, (A  $\wedge$  P x)  $\vee$  (A  $\wedge$  Q x)  $\vee$ 
    (B  $\wedge$  P x)  $\vee$  (B  $\wedge$  Q x).
intros A B P Q H y.
...
H : (A  $\vee$  B)  $\wedge$  (forall x : nat, P x  $\vee$  Q x)
y : nat
=====
A  $\wedge$  P y  $\vee$  A  $\wedge$  Q y  $\vee$  B  $\wedge$  P y  $\vee$  B  $\wedge$  Q y
destruct H as [H1 H2].
...
H1 : A  $\vee$  B
H2 : forall x : nat, P x  $\vee$  Q x
y : nat
...
```

demonstration (continued)

```
...
Q : nat -> Prop
H1 : A  $\vee$  B
H2 : forall x : nat, P x  $\vee$  Q x
destruct H1 as [Ha | Hb].
2 subgoals ...
Q : nat -> Prop
Ha : A
H2 : forall x : nat, P x  $\vee$  Q x
y : nat
=====
A  $\wedge$  P y  $\vee$  A  $\wedge$  Q y  $\vee$  B  $\wedge$  P y  $\vee$  B  $\wedge$  Q y
```

demonstration (continued)

```
destruct (H2 y) as [Hp | Hq].
```

```
3 subgoals
```

```
...
```

```
Ha : A
```

```
Hp : P y
```

```
=====
```

```
A /\ P y \/ A /\ Q y \/ B /\ P y \/ B /\ Q y
```

```
left.
```

```
...
```

```
=====
```

```
A /\ P y
```

```
split.
```

```
4 subgoals
```

```
...
```

```
=====
```

```
A
```

Demonstration (continued)

```
...  
Ha : A  
...  
y : nat  
Hp : P y  
=====
```

A

```
exact Ha.  
...  
=====
```

P y

```
assumption.  
2 subgoals
```

Demonstration (continued)

...

Ha : A

...

Hq : Q y

=====

A /\ P y \/ A /\ Q y \/ B /\ P y \/ B /\ Q y
right; left; split.

...

A /\ Q y

Existential quantification

- ▶ Prove `exists x : T, A x`
 - ▶ You have to find an expression `e` of the right type and prove `A e`

`exists e`

- ▶ Use `H : exists x : T, A x`
`destruct H as [y Hy]` or `case H`.
- ▶ moving from the connective “there exists” to the situation where “there exists” a guy with the right properties

Falsehood and Negation

- ▶ False cannot be proved in the empty context
- ▶ Use `H : False`
`destruct H` or `case H`
 - ▶ Anything can be deduced from False
 - ▶ No new goals
- ▶ Prove $\sim A$
 - ▶ assume A and show there is a contradiction
`intros Ha`
- ▶ Use `H : $\sim A$`
 - ▶ Do this when you know you can prove A
`destruct H` or `case H`

Negation demonstration

```
Lemma example_neg : forall A B : Prop, A -> ~A -> B.  
intros A B Ha Hn.  
Ha : A  
Hn : ~A  
=====  
B  
case Hn.  
Ha : A  
Hn : ~A  
=====  
A
```

Equality

- ▶ Prove $x = x$
`reflexivity`
- ▶ Use $H : \text{forall } x \ y, f \ x \ y = g \ x \ y$
`rewrite H, rewrite <- H, rewrite H in H'`, etc.
 - ▶ find occurrences of $f \ ? \ ?$ in the goal and replace with the corresponding instance of $g \ ? \ ?$
 - ▶ Variables must be guessed, as for `apply`
 - ▶ Variable guessing can be tuned by the user
- ▶ Other approach to using equalities: `injection` to be studied later
- ▶ Other approach to proving equalities: `ring`