# 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 /\ 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 /\ B` to `A` and `B` requires a logical step
- This example was about a *conclusion*, we can have similar problems when `A /\ 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 <= y
H2 : y <= z
====================
 x <= 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 `forall x:T, A x` ?
  - Reason on an arbitrary member of type `T`
  - Arbitrary: we don't know anything about it, it is *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` (here changed for illustration purpose)
  ```

# Implication (in conclusion)

- How do we prove that `A -> 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
- as 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 explicitely consequences using `assert`
- ```
  H : A -> B
  Ha : A
  ================
  C
  assert (H': B).
  ==================
  B
  apply H.
  ==================
  A
  ```
- A second goal has an hypothesis `H'` stating `B`
  ```

# Theorems as functions

- Implication and quantification theorems may be used as functions
- ```
  H : A -> B
  G : forall x : T, D x
  Ha : A
  e :T
  ===============
  C
  assert (H' := H Ha).
  H' : B
  ===============
  C
  assert (G' := G e)
  G' : D e
  ===============
  C
  ```
  ```

# Conjunction

- Prove `A /\ B`
  `split`
- Use `H : A /\ 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 \/ B`
- Choose to prove `A` or to prove `B`
  `left` or `right`
- Use `H : A \/ B`
  `destruct H as [H1 | H2]` or `case H`
  - Two goals generated, one where `A` is given as hypothesis `H1`, one where `B` is given as hypothesis `H2`
  - Need to cover all possibilities
- Some of the tactics have the same name 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]`.
- Ex. :

```
Lemma l1 :  forall A B C, A /\ (B \/ C) ->
         (A /\ B) \/ C.
intros A B C [H1 [H2 | H3]].
```

# Combining tactics

- Use several tactics in one command
- `tac`$_1$`; tac`$_2$,
  `tac`$_2$ is used on all goals generated by `tac`$_1$
- `tac;[tac`$_1$`| ...| tac`$_n$`]`,
  `tac`$_i$ is applied on the $i^{th}$ generated goal

# demonstration

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

# demonstration (continued)

```
...
Q : nat -> Prop
H1 : A \/ B
H2 : forall x : nat, P x \/ Q x
destruct H1 as [Ha | Hb].
2 subgoals ...
Q : nat -> Prop
Ha : A
H2 : forall x : nat, P x \/ Q x
y : nat
=============
A /\ P y \/ A /\ Q y \/ B /\ P y \/ B /\ 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 `exists e`
  - and then prove `A 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 `~A`
  - assume `A` and show there is a contradiction

  `intros Ha`
- Use `H : ~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 : 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`

# Automatic proofs

- `auto`, `tauto`, `intuition`, `trivial` are worth trying for statements of propositional logic.
- `firstorder` is especially suited for proofs that may involve instantiating universal quantifiers (first-order logic).