Proofs about programs

Yves Bertot

September 2019

Performing proofs

- Interactive loop
- Start with `Lemma` *name* `:` *statement*`.` `Proof.`
- Then apply commands that decompose the goal.
- `intros`, `split`, `exists`, `left`, `right`, `auto`, `rewrite`.
- use `apply` with existing theorems.
- Special case for `In`: `simpl` or `compute`
- When finished, save using `Qed.`

Several styles of proof

Several styles of proof

- Navigating logical connectives
- Reasoning about recursive programs
- Reasoning about inductive properties
- Reasoning with decision procedures

Each style calls for a different set of tools

Navigating logical connectives

Use a menomonic table

- Connectives : $\forall$, $\exists$, $\wedge$, $\vee$, $\neg$
- Distinguish between positive and negative occurrences *in conclusion* or *in hypotheses*
- Secondary connective: equivalence.
- Make a special treatment case for equality

Connective to tactic table

	$\rightarrow$	$\forall$	/\	\/
hypothesis H	apply H	apply H	destruct H as [H1 H2]	destruct H as [H1 — H2]
goal	intros H'	intros x	split	left right

	$\exists$	$=$	~
hypothesis H	destruct H as [x Px]	rewrite $\leftarrow$ H rewrite $\rightarrow$ H	exfalso; apply H
goal	exists *e*	reflexivity	intros H'

Worked example

```
Lemma ex0 : ∀ A : Type, ∀ P Q R : A -> Prop, ∀ x : A,
  (∀ z, P z -> Q z) ->
  P x /\ R x -> ∃ y, Q y /\ R y.
Proof.
intros A P Q R x pq pxrx.
1 subgoal
  A : Type
  P, Q, R : A -> Prop
  x : A
  pq : ∀ z : A, P z -> Q z
  pxrx : P x /\ R x
  ============================
  ∃ y : A, Q y /\ R y
```

Worked example

```
destruct pxrx as [px rx].
  x : P x
  rx : R x
  ============================
  ∃ y : A, Q y /\ R y
exists x.

  ============================
  Q x /\ R x
split.
  ============================
  Q x

subgoal 2 is:
 R x
```

Worked example

```
  pq : forall z : A, P z -> Q z
  px : P x
  rx : R x
  ============================
  Q x
apply pq.
  ============================
  P x
exact px.
```

Worked example

```
  A : Type
  P, Q, R : A -> Prop
  x : A
  pq : forall z : A, P z -> Q z
  px : P x
  rx : R x
  ============================
  R x
assumption.
No more subgoals.
Qed.
```

Proofs about recursive functions

- Follow the structure of the function in proofs
  - For recursive programs: induction is probably needed
- pattern-matching constructs: destruct
- Comparison between constructors: discriminate and injection
- Do not forget searching for existing theorems!
- Reshape statements modulo computation: simpl, cbn, replace ... with ...

Example with the sum of first integers

```
Fixpoint sumn (n : nat) :=
  match n with 0 => 0 | S p => sumn p + (p + 1) end.

Lemma sumn_f n : sumn n * 2 = n * (n + 1).
Proof.
induction n as [ | p IH].
2 subgoals

  ============================
  sumn 0 * 2 = 0 * (0 + 1)

subgoal 2 is:
 sumn (S p) * 2 = S p * (S p + 1)
```

Example with the sum of first integers

```
reflexivity.
1 subgoal

  p : nat
  IH : sumn p * 2 = p * (p + 1)
  ============================
  sumn (S p) * 2 = S p * (S p + 1)
cbn [sumn].
  ============================
  (sumn n + (n + 1)) * 2 = S n * (S n + 1)
Search ((_ + _) * _).

Nat.mul_add_distr_r:
   forall n m p : nat, (n + m) * p = n * p + m * p
```

Example with the sum of first integers

```
  ============================
  (sumn n + (n + 1)) * 2 = S n * (S n + 1)
rewrite Nat.mul_add_distr_r.
  IH : sumn p * 2 = p * (p + 1)
  ============================
  sumn n * 2 + (n + 1) * 2 = S n * (S n + 1)
rewrite IH.
  ============================
  n * (n + 1) + (n + 1) * 2 = S n * (S n + 1)
```

Example with the sum of first integers

```
rewrite (Nat.mul_comm (n + 1)), <- Nat.mul_add_distr_r,
  !Nat.add_1_r, Nat.add_succ_r, Nat.add_1_r, Nat.mul_comm.
reflexivity.
Qed.
```

Using advanced tactics

the contents of the previous slide can be replaced by a call to a tactic called ring.

```
ring.
Qed.
```

Using libraries

- Coq comes with existing libraries
- Existing functions have theorems about them
- Important command : `Search`

```
Search filter.
filter_In:
  forall (A : Type) (f : A -> bool) (x : A) (l : list A),
  In x (filter f l) <-> In x l /\ f x = true
```

Examples of powerful libraries

- Mathematical Components
  - Designed for the 4-color theorem, the odd order theorem (Feit-Thompson)
  - Used for elliptic curves, reasoning about robots, combinatorics, transcendance proofs
- Coquelicot
  - Real analysis: limits, derivatives, integrals, mathematical functions
  - Used for numerical computations (wave function, mathematical constants)
- VST
  - Reasoning about programs in C (in connection with Compcert)
  - Used for pointer data structures, security proofs

Proofs about with inductive propositions

- Proofs by induction
- Key insight: inductive structure of proofs
- Elementary bricks: constructors
- Repetition in subproofs (of the same predicate)
- Leads to induction hypotheses

Example proof with inductive propositions

```
Require Import Arith.

Inductive t_closure {T : Type} (R : T -> T -> Prop) :
  T -> T -> Prop :=
  tc1 : forall x y, R x y -> t_closure R x y
| tc_s : forall x y z, R x y -> t_closure R y z ->
  t_closure R x z.

Definition is_suc x y := y = S x.
```

Example proof with inductive propositions

```
Lemma clos_is_suc_lt x y : t_closure is_suc x y -> x < y.
Proof.
induction 1 as [x y base | x y z fst_step tc IH].
  x, y : nat
  base : is_suc x y
  ============================
  x < y

subgoal 2 (ID 35) is:
 x < z
  unfold is_suc in base.
  base : y = S x
  ============================
  x < y
```

Example proof with inductive propositions

```
rewrite base.
============================
x < S x

now apply Nat.lt_succ_diag_r.
```

Example proof with inductive propositions

```
  x, y, z : nat
  fst_step : is_suc x y
  tc : t_closure is_suc y z
  IH : y < z
  ============================
  x < z
apply Nat.lt_trans with y.
  unfold is_suc in fst_step.
  rewrite fst_step.
  now apply Nat.lt_succ_diag_r.
exact IH.
Qed.
```

inversion

- An inductive type may have a constructor of the form `A -> B`
- If no other constructor can prove `B`
- Any instance of `B` can only hold if `A` hold
- In practice, it feels like the implication is inverted

Inversion example

```
Inductive i_even : nat -> Prop :=
  | ie0 : i_even 0
  | ie2 : forall m, i_even m -> i_even (S (S m)).
```

- A statement of the form `i_even (S (S (S x)))` cannot have been proved using constructor ie0.
- constructor `ie2` was used in such a way that `m` was `S x`
- For the proof to be complete, `i_even m` was proved
- So we can deduce `i_even (S x)`
- Same reasoning to show that `i_even 1` is false.

Inversion example

```
Lemma i_even_inv5 : i_even 5 -> False.
Proof.
intros ev5.
inversion ev5 as [|n3 ev3 eq3].
  ev5 : i_even 5
  n3 : nat
  ev3 : i_even 3
  eq3 : n3 = 3
  ============================
  False
inversion ev3 as [|n1 ev1 eq1].
  ev1 : i_even 1
  eq1 : n1 = 1
  ============================
  False
```

Inversion example

```
  ev1 : i_even 1
  eq1 : n1 = 1
  ============================
  False
inversion ev1.
No more subgoals.
Qed.
```

Proofs by reflection

- Reason on algorithms that perform proofs
- Use the proved algorithms to perform proofs
- Three stages
  - Find a generic language to describe problems
  - Write proved programs about the language
  - Make Coq recognize instances
- Example material: proofs by associativity (+ commutativity)