Induction over natural numbers

Yves Bertot

August 2011

Useful tools to perform proofs about natural numbers

- To work comfortably with natural numbers, it is better to load known facts and decision procedures
  - Require Import Arith Omega.
- Decision procedures
  - ring proves equalities between polynomial formulas does not use hypotheses
  - omega proves systems of linear inequalities uses hypotheses
- It takes time learning to recognize linear inequalities

Example with ring

Lemma example : forall n m, 2*n*(n + S m) = 2*n + n*n + 2*n*m + n*n.
intros n m.
n : nat
m : nat
==============
2*n*(n + S m) = 2*n + n*n + 2*n*m + n*n
ring.
Proof completed.

Example with omega

Lemma example : forall n m, 2 < n -> 2*m - 3 < n -> m < n.
intros n m H2 H2m3.
n : nat
m : nat
H2 : 2 < n
H2m3 : 2*m - 3 < n
==============
m < n
omega.
Proof completed.

Mathematical principle of induction over natural numbers

- Any property that is true for 0 and can be proved for any number n+1 assuming that it is already true for n is true for all natural numbers
- The assumption is called an induction hypothesis
- This is usually called the principle of induction
- Proofs by induction concerning natural numbers are frequent in Coq
- A good model of other proofs by induction

Induction principle in Coq

- The principle of induction can be written in Coq
  - forall P : nat -> Prop,
    P O -> (forall p:nat, P n -> P (S p)) ->
    forall n : nat, P n
- This principle exists in Coq, with the name nat_ind
- This principle is naturally used by tactics elim and induction
**Induction tactic**

- Written as induction $e$
- Finds instances of $e$ in the current goal
  - makes the goal appear as $P e$
- Generates two goals
  - In the first goal, one must prove $P 0$
  - In the second goal, a new $n$ is provided
    - one must prove $P (Sn)$, assuming $P n$
    - the new assumption is usually called $IHn$

**Useful facts for the next example**

<table>
<thead>
<tr>
<th>SearchRewrite $(0 + _)$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>plus_O_n : forall n : nat, 0 + n = n.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SearchRewrite $((S _) + _)$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>plus_Sn_m : forall n m : nat, S n + m = S (n + m)</td>
</tr>
</tbody>
</table>

**Example with the induction tactic**

**Lemma** plus_n_0 : forall n, n + 0 = n.

```plaintext
forall n : nat, n + 0 = n
```

```plaintext
intros n.
```

```plaintext
n : nat
```

```plaintext
forall n : nat, n + 0 = n
```

```plaintext
 induction n.
```

```plaintext
2 subgoals
```

```plaintext
0 + 0 = 0
```

```plaintext
subgoal 2 is:
```

```plaintext
S n + 0 = S n
```

**Example induction (continued)**

- The induction principle was used
- The property $P$ is $fun \ y : \ nat \Rightarrow y + 0 = y$
- In the first goal, applied to $0$
- In the conclusion of the second goal, applied to $S n$
- The first goal can be solved by applying plus_O_n

```plaintext
0 + 0 = 0
```

```plaintext
apply plus_O_n. (* The first goal disappears. *)
```

```plaintext
n : nat
```

```plaintext
IHN : n + 0 = n
```

```plaintext
S n + 0 = S n
```

```plaintext
rewrite plus_Sn_m.
```

```plaintext
...
```

```plaintext
S (n + 0) = S n
```

```plaintext
rewrite IHN.
```

```plaintext
S n = S n
```

```plaintext
reflexivity. (* This finishes the proof. *)
```

**Reasoning about recursive functions**

- Using the tactic reflexivity, one can solve any goal
  - if $f x = e$
    - if $f x$ and $e$ are the same by definition
- Example

```plaintext
Print pred.
```

```plaintext
pred = fun n => match n with 0 => n | S u => u end
```

```plaintext
(* reflexivity can be used on the following goal: *)
```

```plaintext
reflexivity.
```
> Using the tactic `change` one can replace `f x` with `e` if `f x` and `e` are the same by definition.

▶ Example

======
```latex
\text{pred (S n) = n}
```

▶ To avoid providing the result expression one can use `unfold` or `simpl`.

======
```latex
\text{pred (S n) = n}
```

▶ Example with `simpl`.

Print `plus`.

```
\text{Print plus.}
```

```
\text{plus = fix plus (n m : nat) : nat :=}
\text{match n with 0 => m | S p => S (plus p m) end}
```

(* on the following goal, use `simpl` *)

```
\text{S x + y = S (x + y)}
```

```
\text{simpl.}
```

```
\text{S (x + y) = S (x + y)}
```

▶ A proof by induction is a `squeeze`.

▶ Usually easier to prove weaker statements.

▶ But in induction, if the statement is weak, so is the induction hypothesis.

▶ So in induction, sometimes easier to prove stronger statement.

▶ Stronger statements usually found by have more universal quantifications.

Example stronger induction.

Fixpoint `div2` `n` :=

```
\text{match n with S (S p) => S (div2 p) | _ => 0 end.}
```

Fixpoint `mod2` `n` :=

```
\text{match n with S (S p) => mod2 p | _ => n end.}
```

▶ `div2` and `mod2` actually have three cases: 0, 1, and `S (S p)`.

Lemma `mod2div2Q` : `forall` `n`, `n` = `2 * div2 n + mod2 n`.

```
\text{intros n.}
```

```
\text{assert (H : n = 2 * div2 n + mod2 n |
\text{S n = 2 * div2 (S n) + mod2 (S n)).}
```

```
\text{induction n.}
```

```
\text{0 = 2 * div2 0 + mod2 0 | 1 = 2 * div2 1 + mod2 1}
```

```
\text{simpl.}
```

```
\text{0 = 0 | 1 = 1}
```

auto.
Proof on div2 (continued)

IHn : n = 2 * div2 n + mod2 n /
S n = 2 * div2 (S n) + mod2 (S n)
============================
S n = 2 * div2 (S n) + mod2 (S n)

S (S n) = 2 * div2 (S (S n)) + mod2 (S (S n))
destruct IHn as [I1 _].
I1 : n = 2 * div2 n + mod2 n
============================
S (S n) = 2 * div2 (S (S n)) + mod2 (S (S n))
simpl.
S (S n) = S (div2 n + S (div2 n + 0) + mod2 n)

Proof on div2 (continued)

replace (S (div2 n + S (div2 n + 0) + mod2 n)) with
(S (S (2 * div2 n + mod2 n))) by ring.
I1 : n = 2 * div2 n + mod2 n
============================
S (S n) = S (S (2 * div2 n + mod2 n))
rewrite <- I1; reflexivity.
(* At this point we finished proving the strong statement. *)
n : nat
H : n = 2 * div2 n + mod2 n /
S n = 2 * div2 (S n) + mod2 (S n)
============================
n = 2 * div2 n + mod2 n
intuition.

Proofs by induction and universal quantification

▶ A common error: intros and induction
▶ Too much introduction blocks a specific value
▶ Proof by induction sometimes require more general statement
▶ Two solutions:
  ▶ be more prudent with intros in the first place
  ▶ use revert and generalize to make more general statements

Example: induction and general statement

Fixpoint add n m :=
match n with 0 => m | S p => add p (S m) end.

Lemma addnS : forall n m, add n (S m) = S (add n m).
intros n m.
================
add n (S m) = S (add n m)
induction n; [reflexivity | ].

Induction statement not general enough

IHn : add n (S m) = S (add n m)
============================
add (S n) (S m) = S (add (S n) m)
simpl.
============================
add n (S (S m)) = S (add n (S m))
rewrite IHn.
============================
add n (S (S m)) = S (add n (S m))

▶ This situation is stuck: add n (S (S m))

Solution: start again, use intros with moderation

Lemma addnS : forall n m, add n (S m) = S (add n m).
intros n.
============================
forall m, add n (S m) = S (add n m)
induction n; [reflexivity | ].
IHn : forall m : nat, add n (S m) = S (add n m)
============================
forall m, add n (S (S m)) = S (add n m)
intros m; simpl.
IHn : forall m : nat, add n (S m) = S (add n m)
m : nat

apply IHn.
Proof completed.
Qed.

Last advice

- When performing proofs about a recursive function
  - find what arguments decrease at each call
  - reason by induction on one of these arguments
  - have the other ones universally quantified, if possible
- Use the `simpl` and `change` tactics to reason about function execution
- Use the `case` tactic when `match ... with` appears in the goal