Chapter 1

Library clodomir_induction

Capítulo 2 - Proof by Induction (Induction)

Exemplos induction

`Theorem` *plus_x_O* : ∀ *x*:*nat*,
x = *x* + 0.

`Theorem` *minus_diag* : ∀ *x*,
x - *x* = 0.

Exercise: 2 stars, standard, recommended (basic_induction)

`Theorem` *mult_0_r* : ∀ *x*:*nat*,
x × 0 = 0.

`Theorem` *plus_x_Sy* : ∀ *x* *y* : *nat*,
S (*x* + *y*) = *x* + (*S* *y*).

`Theorem` *plus_comm* : ∀ *x* *y* : *nat*,
x + *y* = *y* + *x*.

`Theorem` *plus_assoc* : ∀ *x* *y* *z* : *nat*,
x + (*y* + *z*) = (*x* + *y*) + *z*.

Exercise: 2 stars, standard (double_plus)

```
Fixpoint double (x:nat) :=
  match x with
  | O ⇒ O
  | S x' ⇒ S (S (double x'))
  end.
```

`Lemma` *double_plus* : ∀ *x*, *double* *x* = *x* + *x* .

Exercise: 2 stars, standard, optional (evenb_S)

`Theorem` *evenb_S* : ∀ *x* : *nat*,
evenb (*S* *x*) = *negb* (*evenb* *x*).

Exemplo assert

`Theorem` *mult_0_plus'* : $\forall\ x\ y : nat,$
$(0 + x) \times y = x \times y.$

`Theorem` *plus_rearrange* : $\forall\ x\ y\ z\ t : nat,$
$(x + y) + (z + t) = (y + x) + (z + t).$

`Theorem` *plus_assoc'* : $\forall\ x\ y\ z : nat,$
$x + (y + z) = (x + y) + z.$

`Theorem` *plus_assoc''* : $\forall\ x\ y\ z : nat,$
$x + (y + z) = (x + y) + z.$

Exercise: 2 stars, advanced, recommended (plus_comm_informal)
Translate your solution for *plus_comm* into an informal proof:
Theorem: Addition is commutative.
Theorem: For any n and m, n + m = m + n
Proof : By induction on n.

- First, suppose $n = 0$. We must show: 0 + m = m + 0, it follows from the definition of +.
- Next, suppose $n = S\ n'$, with n' + m = m + n', then S n' + m = m + S n' – apply + and plus_n_Sm S (n' + m) = S (m + n') – apply hypotesis S (m + n') = S (m + n').

Qed.

`Theorem` *plus_swap* : $\forall\ x\ y\ z : nat,$
$x + (y + z) = y + (x + z).$

`Theorem` *mult_x_0* : $\forall\ x : nat,$
$x \times 0 = 0.$

`Theorem` *mult_x_Sy*: $\forall\ x\ y : nat,$
$x \times S\ y = x + x \times y.$

`Theorem` *mult_comm* : $\forall\ x\ y : nat,$
$x \times y = y \times x.$

`Check` *leb*.

`Theorem` *leb_refl* : $\forall\ x$:*nat*,
true = *leb x x*.

`Theorem` *zero_neqb_S* : $\forall\ x : nat,$
eqb 0 ($S\ x$) = *false*.

`Theorem` *andb_false_r* : $\forall\ b : bool,$
andb b false = *false*.

`Theorem` *plus_leb_compat_l* : $\forall\ x\ y\ z : nat,$
leb x y = *true* $\rightarrow$ *leb* ($z + x$) ($z + y$) = *true*.

```
Theorem S_neqb_0 : ∀ x : nat,
  eqb (S x) 0 = false.

Theorem mult_1_l : ∀ x : nat,
  1 × x = x.

Theorem all3_spec : ∀ b c : bool,
    orb (andb b c) (orb (negb b) (negb c)) = true.

Theorem mult_plus_distr_r : ∀ x y z : nat,
  (x + y) × z = (x × z) + (y × z).

Theorem mult_assoc : ∀ x y z : nat,
  x × (y × z) = (x × y) × z.

Theorem eqb_refl : ∀ x : nat,
  true = eqb x x.

Theorem plus_swap' : ∀ x y z : nat,
  x + (y + z) = y + (x + z).

Module binnats.

  Inductive bin : Set :=
  | O : bin
  | D : bin → bin
  | P : bin → bin.

  Fixpoint bininc (b:bin) : bin :=
    match b with
      | O ⇒ P O
      | D b' ⇒ P b'
      | P b' ⇒ D (bininc b')
    end.

  Fixpoint bin2nat (b:bin) : nat :=
    match b with
      | O ⇒ 0
      | D b' ⇒ double (bin2nat b')
      | P b' ⇒ S (double (bin2nat b'))
    end.

  Theorem bin2nat_bininc_comm : ∀ b:bin,
    bin2nat (bininc b) = S (bin2nat b).

    Fixpoint nat2bin (n:nat) : bin :=
      match n with
        | 0 ⇒ O
        | S n' ⇒ bininc (nat2bin n')
      end.

    Theorem nat2bin2nat_id : ∀ n:nat,
```

```
    bin2nat (nat2bin n) = n.

  Eval simpl in (nat2bin (bin2nat O)).
  Eval simpl in (nat2bin (bin2nat (D (D (D O))))).

  Fixpoint normalize (b:bin) : bin :=
    match b with
      | O ⇒ O
      | D b' ⇒ match normalize b' with
                 | O ⇒ O
                 | nb ⇒ D nb
               end
      | P b' ⇒ P (normalize b')
    end.

Definition imp_id_proof := fun (p:Prop) ⇒ (fun (d:p) ⇒ d).

Check imp_id_proof (3=3).
Check imp_id_proof (3=3) (refl_equal 3).

Theorem imp_id : ∀ P:Prop, P → P.

Print imp_id.
```

Eval simpl in bininc (normalize (D (P (D O)))) = (normalize (P (P (D O)))).

```
Definition bindouble (b:bin) : bin :=
  match b with
    | O ⇒ O
    | D n' ⇒ D (D n')
    | P n' ⇒ D (P n')
  end.

Lemma bininc_twice : ∀ b:bin,
  bininc (bininc (bindouble b)) = bindouble (bininc b).

Lemma double_bindouble : ∀ n:nat,
  nat2bin (double n) = (bindouble (nat2bin n)).

Lemma bininc_bindouble: ∀ b:bin,
  bininc (bindouble b) = P b.

Theorem bin2nat2bin_n_eq_norm_n : ∀ b:bin,
  nat2bin (bin2nat b) = normalize b.

End binnats.
```