```coq
(* Preparation a l'examen d'INF 471 (partie Coq)


 les definitions nouvelles se trouvent vers la fin du fichier
   (predicat Even)

***************************************************************)

(* Etude d'une axiomatique de l'arithmetique (reprise du projet)  *)

(**************************************************

 Note : ceci est une illustration de notions vues en cours. En Coq
  les entiers naturels, l'addition et la multiplications sont traitÃ©es
  sans axiomes et d'une maniere beaucoup plus efficace, ainsi que les
   inégalités <= et <


 De mÃªme, ce fichier utilise volontairement des tactiques 'de base'.
 Il peut sans nul doute Ãªtre beaucoup plus concis.


********************************************************)

Parameter Nat : Set.

Parameter zero : Nat.

Parameter S : Nat -> Nat.

Parameter plus : Nat -> Nat -> Nat.

Parameter mult : Nat -> Nat -> Nat.

Notation "n + p" := (plus n p).

Notation "n * p" := (mult n p).

(* Axiomes (sont des théorèmes dans la bibliotheque standard de Coq) *)

Axiom zero_not_S : forall n:Nat, S n <> zero.

Axiom S_injective : forall n p, S n = S p -> n = p.

Axiom zero_plus_n : forall n:Nat, zero + n = n.

Axiom S_plus_n : forall n p:Nat, S n + p = S (n + p).


Axiom zero_mult_n : forall n:Nat, zero * n = zero.

Axiom S_mult_n : forall n p: Nat, (S n)*p = p + (n * p).

(* axiome de recurrence *)

Axiom Nat_ind : forall (P:Nat->Prop), P zero ->
                                       (forall p:Nat, P p -> P (S p)) ->
                                       forall n:Nat, P n.

Hint Resolve zero_not_S S_injective zero_plus_n S_plus_n.

Lemma fib_ind : forall (P:Nat->Prop),
                P zero -> P (S zero) ->
                (forall p : Nat, P p -> P (S (S p))) ->
                forall n : Nat, P n.
Proof.
 intros P H0 H1 H n.
 assert (P n /\ P (S n)).
destruct n using Nat_ind.
 auto.
 destruct IHn;auto.
 destruct H2.
 auto.
 Qed.

Lemma zero_or_S : forall n:Nat, n=zero \/ exists p:Nat, n = S p.
Proof.
 intro n; induction n using Nat_ind.
 auto.
 right;exists n;auto.
Qed.


(* Le prédicat 'inférieur ou égal' est axiomatisé, à la différence
   du cours et de la bibliothèque standard de Coq (où il est défini) *)

Parameter Le : Nat -> Nat -> Prop.

Notation "n <= p" := (Le n p).

Axiom Le_n : forall n:Nat, n<=n.

Axiom Le_S : forall n p:Nat, n <= p -> n <= S p.

(* Permet aux tactiques "auto" et "auto" d'utiliser ces axiomes *)

Hint Resolve Le_n Le_S.


(* Le_ind exprime la "récurrence complète":
   Pour prouver que   n <= q  -> P q
   il suffit de prouver P n et que
                      si n <= p et P p, alors P(S p)
*)


Axiom Le_ind : forall n:Nat, forall P:Nat -> Prop,
               P n ->
               (forall p : Nat, n <= p -> P p -> P (S p)) ->
               forall q:Nat, n<= q -> P q.

(* Cette preuve utilise Le_ind *)
Lemma Le_Sn_p : forall n p, S n <= p -> n <= p.
Proof.
 intros n p H.
 induction H using Le_ind.
 auto.
 auto.
Qed.

(* on place Ce résultat dans la base de théorèmes pour auto *)

Hint Resolve Le_Sn_p.


Theorem Le_trans : forall n p q:Nat, n <=  p  -> p <= q -> n <= q.
Proof.
 intros n p q H;induction  H using Le_ind;auto.
Qed.


Lemma Le_inv : forall n p : Nat, n <= p -> n= p \/
                                            exists q:Nat, p = S q /\ n <= q.
Proof.
 intros n p H;induction H using Le_ind.
 auto.
 destruct IHLe.
 right;exists p;auto.
 destruct H0.
 destruct H0.
 right;exists (S x).
 split.
 rewrite H0;trivial.
 auto.
Qed.

Lemma Le_n_zero : forall n:Nat, n<=zero -> n=zero.
Proof.
 intros n H.
 (*
  1 subgoal

  n : Nat
  H : n <= zero
  ============================
   n = zero

  Ici, on veut appliquer Le_inv
  Cette fonction prend trois arguments : un entier n
                                         un entier p
                                         une preuve H : n<= p

  Si on veut faire une preuve par cas, il suffit d'utiliser destruct
  sur le terme (Le_inv n zero H)
  dont le type est
  n= zero \/  exists q:Nat, zero = S q /\ n <= q.
*)

 destruct (Le_inv n zero H).
 auto.
 destruct H0.
 destruct H0.
 destruct (zero_not_S x).
 rewrite H0;auto.
Qed.



Lemma Le_n_Sn : forall n, n <= S n.
Proof.
 info auto.
 (*
   intro n; simple apply Le_S; simple apply Le_n.
 *)
Qed.



Lemma Le_zero_n : forall n:Nat, zero <= n.
Proof.
 (* on peut proceder par recurrence sur n *)

 intro n;induction n using Nat_ind.
 (*

  cas de base :
 ============================
   zero <= zero
 *)
 apply Le_n.

 (* pas de recurrence

  1 subgoal

  n : Nat
  IHn : zero <= n
  ============================
   zero <= S n

 *)

 apply Le_S;assumption.
Qed.

(* en fait, cette preuve pouvais se faire en une ligne :

Lemma Le_zero_n : forall n:Nat, zero <= n.
Proof.
 intro n; induction n using Nat_ind;auto.
Qed.
 *)


Hint Resolve Le_zero_n.


Definition Lt n p := S n <= p.
Notation "n < p" := (Lt n p).

(* A tout moment, on peut transformer un but contenant (a < b)
```

```
  par 'unfold lt' (qui le remplace par ((S a) <= b)

   On peut faire  ce remplacement dans une hypothèse H avec
  'unfold lt in H'

*)



 Lemma Lt_trans : forall n p q:Nat, n < p -> p < q -> n < q.
Proof.
  unfold Lt;intros n p q H H0.

 (*
1 subgoal

  n : Nat
  p : Nat
  q : Nat
  H : S n <= p
  H0 : S p <= q
  ============================
   S n <= q

 *)
  apply Le_trans with p;auto.
 Qed.



Lemma Le_cases : forall n p, n <= p -> n < p \/ n = p.
Proof.
 (* Ici, on choisit la "recurrence a partir de n (Le_ind) *)
 intros n p H; induction H using Le_ind;auto.
 (*
1 subgoal

  n : Nat
  p : Nat
  H : n <= p
  IHLe : n < p \/ n = p
  ============================
   n < S p \/ n = S p

  *)

  (* L'hypothese de recurrence est une disjonction.
  dans le premier cas la transitivite de Lt sera faicle a utiliser.
  dans le second, si on remplace n par p, il sera facile de prouver p < S p
  *)

 destruct IHLe.
 left; eapply Lt_trans with p;auto.

 unfold Lt;auto.
 left; rewrite H0;unfold Lt;auto.
Qed.
```

```
Lemma Lt_Le : forall n p, n < p -> n <= p.
Proof.
 intros n p H.
 auto.
Qed.


Lemma Le_S_S : forall n p, n <= p -> S n <= S p.
Proof.
(* La encore, il s'agit d'une recurrence "a partir de n" *)
 intros n p H;induction H using Le_ind; info auto.
Qed.


Lemma Le_n_S_p : forall n p, n <= S p -> n = S p \/ n <= p.
Proof.

 (* Le_inv donne deux cas pour l'hypothese n <= S p :
    a) n = S P    (preuve triviale
    b) il existe q tel que S p = S q et n <= q *)
 intros n p H; destruct (Le_inv _ _ H).
 auto.
 destruct H0.
 destruct H0.
 (*
  n : Nat
  p : Nat
  H : n <= S p
  x : Nat
  H0 : S p = S x
  H1 : n <= x
  ============================
   n = S p \/ n <= p

 *)
 assert (H2 : p = x).
 apply S_injective; assumption.
 rewrite H2;auto.
Qed.

Lemma Le_S_SR : forall n p, S n <= S p -> n <= p.
Proof.
 (* rien de nouveau ... *)
 intros n p H; destruct (Le_n_S_p _ _ H).
 assert (H1 : n=p).
 apply S_injective;assumption.

 rewrite H1;auto.
 (*
1 subgoal

  n : Nat
  p : Nat
  H : S n <= S p
  H0 : S n <= p
  ============================
   n <= p
 *)
```

```
 apply Le_trans with (S n);auto.
Qed.

Lemma not_le_Sn_n : forall n:Nat,~ S n <= n.
Proof.
 intro n.
 induction n using Nat_ind.
 intro.
 destruct (Le_inv _ _ H).
 destruct (zero_not_S zero);auto.
 destruct H0.
 destruct H0.
 destruct (zero_not_S x);auto.
 intro H;destruct IHn.
 apply Le_S_SR.
 assumption.
Qed.


Lemma Lt_irreflexive : forall n, ~ n < n.
Proof.
 unfold Lt.
 apply not_le_Sn_n.
Qed.


Lemma not_lt_S_n_n : forall n, ~ (S n < n).
Proof.
 intros n H.
 destruct (not_le_Sn_n n).
 auto.
Qed.


Lemma Le_antisym : forall n p, n <= p -> p <= n -> n = p.
Proof.
 (* Apparemment, c'est cette preuve qui a le plus fait souffrir ... *)

 intro n; induction n using Nat_ind.
 intros p Hp Hp'.
 symmetry;apply Le_n_zero.
 assumption.
 intros p H0 H1.
 destruct (Le_cases _ _ H0);destruct (Le_cases _ _ H1);auto.

 destruct (not_lt_S_n_n (S n)).
 unfold Lt.
 apply Le_S_S.
 apply Le_trans with p;auto.
 apply Le_S_SR.
 trivial.
Qed.


(* Proprietes de l'addition *)

Lemma n_plus_zero : forall n:Nat, n + zero =n.
Proof.
```

```
 intro n; induction n using Nat_ind.
 auto.
 rewrite S_plus_n.
 rewrite IHn;trivial.
Qed.



(* La commande suivante permet d'utiliser les égalités ci-dessous
   avec la commande 'autorewrite with peano'
  Attention aux bouclages éventuels !!!!!!!
   *)

Hint Rewrite n_plus_zero zero_plus_n S_plus_n: peano.

Lemma n_plus_S : forall n p:Nat, n+ S p = S(n+p).
Proof.
 intro n; induction n using Nat_ind.
 intro p.
 autorewrite with peano;trivial.
 intros p .
 autorewrite with peano.
 rewrite IHn.
 trivial.
Qed.

Hint Rewrite n_plus_S :peano.


Hint Resolve n_plus_S n_plus_zero.

Theorem plus_comm : forall n p:Nat, n + p = p + n.
Proof.
 intro n; induction n using Nat_ind.
 intro; autorewrite with peano;trivial.
 intros p .
 transitivity (S (n + p));auto.
 rewrite IHn.
 auto.
Qed.


Hint Rewrite zero_mult_n S_mult_n : peano.

Theorem plus_assoc : forall n p q:Nat, (n+p)+q = n+(p+q).
Proof.
 intro n.
 induction n using Nat_ind.
 intros;autorewrite with peano.
 trivial.
 intros p q.
  autorewrite with peano.
  rewrite IHn;auto.
Qed.

Hint Rewrite plus_assoc :peano.
```

```
Lemma plus_zero : forall n p:Nat, n+p=zero -> n=zero /\ p=zero.
Proof.
 intro n; induction n using Nat_ind.
 auto.
 intros.
 autorewrite with peano in H.
 split;auto.
intros p H.
 autorewrite with peano in H.
 destruct (zero_not_S (n + p)).
 auto.
Qed.

Theorem mult_n_zero : forall n:Nat, n* zero = zero. (* A faire *)
Proof.
  intro n; induction n using Nat_ind.
  autorewrite with peano.
  trivial.
  autorewrite with peano.
  trivial.
Qed.

Hint Rewrite mult_n_zero : peano.




Theorem mult_n_Sp : forall n p:Nat, n * (S p) = n + (n * p).
Proof.
  intro n; induction n using Nat_ind.
  intros;autorewrite with peano.
  trivial.
  intros p;autorewrite with peano.
  rewrite IHn.

 (* ici, autorewrite ne permet pas d'automatiser la fin de la preuve *)
 rewrite <- plus_assoc.
 rewrite <- plus_assoc.


(* On  veut utiliser la commutativite de l'addition pour remplacer
   p0 + p par p + p0
  Pour specifier à rewrite ces arguments, on les donne comme arguments
  de plus_comm *)

 rewrite (plus_comm p n).
 trivial.
Qed.

Hint Rewrite mult_n_Sp:peano.


Lemma mult_comm: forall n p:Nat, n*p = p*n.
Proof.
(* Comme pour l'addition, on procede par recurrence sur n *)

 intro n; induction n using Nat_ind;intros;autorewrite with peano.
```

```
 trivial.
 (*
1 subgoal

  n : Nat
  IHn : forall p : Nat, n * p = p * n
  p : Nat
  ============================
   p + n * p = p + p * n

*)
 rewrite (IHn p).
 trivial.
Qed.


Lemma one_mult_n : forall n, (S zero)*n = n.
Proof.
 (* meme pas besoin de recurrence *)
  intro n; autorewrite with peano.
 trivial.
Qed.

(* A faire *)

Lemma n_mult_one : forall n, n*(S zero) = n.
Proof.
  intro n; autorewrite with peano.
 trivial.
Qed.



Lemma mult_zero : forall n p:Nat, n*p=zero -> n=zero \/ p=zero.
Proof.
 (* strategie, une recurrence sur n, qui aura un case de base facile *)
 intro n; induction n using Nat_ind;auto.
 (*
  passons aux choses serieuses :

  n : Nat
  IHn : forall p : Nat, n * p = zero -> n = zero \/ p = zero
  ============================
   forall p : Nat, S n * p = zero -> S n = zero \/ p = zero
 *)
 intros p H.

 (*
  1 subgoal

  n : Nat
  IHn : forall p : Nat, n * p = zero -> n = zero \/ p = zero
  p : Nat
  H : S n * p = zero
  ============================
   S n = zero \/ p = zero

  Dans H, il suffit de faire apparaitre une addition.
```

```
*)

 autorewrite with peano in H.
(*

1 subgoal

  n : Nat
  IHn : forall p : Nat, n * p = zero -> n = zero \/ p = zero
  p : Nat
  H : p + n * p = zero
  ============================
   S n = zero \/ p = zero


*)
 generalize (plus_zero p (n * p) H).

(*

1 subgoal

  n : Nat
  IHn : forall p : Nat, n * p = zero -> n = zero \/ p = zero
  p : Nat
  H : p + n * p = zero
  ============================
   p = zero /\ n * p = zero -> S n = zero \/ p = zero

*)
tauto.
Qed.



Lemma Le_plus : forall n p:Nat, n <= p -> exists q:Nat, p = n+q.
Proof.
 (* La encore, induction a partir de n *)
 intros n p H.
 induction H using Le_ind.

 (*  cas de base:

 n : Nat
  ============================
   exists q : Nat, n = n + q
 *)
exists zero; autorewrite with peano;  trivial.
 (*
  pas de recurrence :

  1 subgoal

  n : Nat
  p : Nat
  H : n <= p
  IHLe : exists q : Nat, p = n + q
```

```
  ============================
   exists q : Nat, S p = n + q
 *)

 destruct IHLe.
 exists (S x).
 (*
1 subgoal

  n : Nat
  p : Nat
  H : n <= p
  x : Nat
  H0 : p = n + x
  ============================
   S p = n + S x
*)
 autorewrite with peano.
 rewrite H0;auto.
Qed .



Theorem plus_le : forall n p, n <= n + p.
Proof.
 (* choix startegique; une recurrence sur p *)
 intros n p; induction p using Nat_ind.
 autorewrite with peano;auto.
 (*
1 subgoal

  n : Nat
  p : Nat
  IHp : n <= n + p
  ============================
   n <= n + S p
 *)
 apply Le_trans with (n + p).
 auto.
 autorewrite with peano.
 auto.

Qed.



(* Debut des nouvelles definitions *)

Parameter Even : Nat -> Prop.

Axiom Even_zero : Even zero.

Axiom Even_S_S : forall n:Nat, Even n -> Even (S (S n)).

Axiom Even_ind : forall (P:Nat->Prop),
   P zero ->
   (forall n :  Nat, Even n -> P n -> P (S (S n))) ->
   forall n:Nat, Even n -> P n.
```

```
Definition Odd n := Even (S n).


Lemma Even_inv : forall n, Even n ->
                  n=zero \/ exists q:Nat, n=S (S q) /\ Even q.
Proof.
Admitted.

Lemma not_Even_1 : ~ Even (S zero).
Proof.
 intro H.
 destruct (Even_inv _ H).
Admitted.


Lemma Even_S_S_inv : forall n:Nat, Even (S (S n)) -> Even n.
Admitted.

Theorem Odd_ind : forall (P :Nat-> Prop),
    P (S zero) ->
    (forall n:Nat, Odd n -> P n -> P (S (S n))) ->
    forall n : Nat, Odd n -> P n.
Proof.
 unfold Odd.
 intros P H1 HS n.
 induction n using fib_ind.
Admitted.




Lemma Even_plus_Odd : forall n , Even n -> forall p, Odd p -> Odd (p + n).
Proof.
 intros n H p Hp;induction Hp using Odd_ind.
Admitted.
```