Tipos Inductivos

- El tipo `A /\ B`

Reglas de introducción

```
Inductive and (A : Prop) (B : Prop) : Prop :=  conj : A -> B -> A /\ B
For conj: Arguments A, B are implicit
For and: Argument scopes are [type_scope type_scope]
For conj: Argument scopes are [type_scope type_scope _ _]

Coq < Check conj.
conj : forall A B : Prop, A -> B -> A /\ B
```

Reglas de eliminación

```
and_ind : forall A B P : Prop, (A -> B -> P) -> A /\ B -> P
```

Reglas de cómputo

```
Coq < Parameters A B P:Prop.
A is assumed
B is assumed
P is assumed

Coq < Parameters (a:A) (b:B).
a is assumed
b is assumed

Coq < Parameter f:A->B->P.
f is assumed
```

```
Coq < Eval simpl in (and_ind f (conj a b)).
     = f a b
     : P
```

- El tipo `nat` *Reglas de introducción*

```
Inductive nat : Set :=  O : nat | S : nat -> nat
For S: Argument scope is [nat_scope]

Coq < Check S.
S
     : nat -> nat
```

Reglas de eliminación

```
 Coq < Check nat_ind.
nat_ind
     : forall P : nat -> Prop,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n

Coq < Check nat_rec.
nat_rec
     : forall P : nat -> Set,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n

Coq < Check nat_rect.
nat_rect
     : forall P : nat -> Type,
       P 0 -> (forall n : nat, P n -> P (S n)) -> forall n : nat, P n
```

Reglas de cómputo (algunas)

```
Coq < Lemma nat_ind_1:forall (P0 : nat -> Prop) (init : P0 0)
     (paso : forall n : nat, P0 n -> P0 (S n)), nat_ind P0 init paso 0 = init
```

```
1 subgoal

  ============================
   forall (P : nat -> Prop) (init : P 0)
     (paso : forall n : nat, P n -> P (S n)), nat_ind P init paso 0 = init

nat_ind_1 < auto.
Proof completed.

Lemma nat_rec_2: forall (P : nat -> Set) (init : P 0) (paso : forall n : nat, P n -> P (S n))
  (n : nat), nat_rec P init paso (S n) = paso n (nat_rec P init paso n).
auto.
Save.
```

- **El tipo de las listas de elementos de un conjunto**
  Reglas de introducción

```
Inductive elist : Type :=
enil: elist
|econs:forall A:Set, A->elist->elist.

Implicit Arguments econs [A].

Check (econs true (econs 3 enil)).


Inductive list (A : Set) : Set :=
    nil : list A | cons : A -> list A -> list A
For nil: Argument A is implicit
For cons: Argument A is implicit
For list: Argument scope is [type_scope]
For nil: Argument scope is [type_scope]
For cons: Argument scopes are [type_scope _ _]


Coq < Require Export List.

Coq < Check nil.
Toplevel input, characters 27-30
> Check nil.
>       ^^^
Error: Cannot infer an instance for the implicit parameter A of nil

Coq < Check nil (A:=nat).
```

```
nil
     : list nat

Coq < Check cons.
cons
     : forall A : Set, A -> list A -> list A

Coq < Print implicit list.
Toplevel input, characters 79-83
> Print implicit list.
>                ^^^^
Syntax error: '.' expected after [vernac:command] (in [vernac:Vernac_.vernac])

Coq < Print Implicit list.
list : Set -> Set

No implicit arguments

Coq < Print Implicit cons.
cons : forall A : Set, A -> list A -> list A

Argument A is implicit

Coq < Print Implicit nil.
nil : forall A : Set, list A

Argument A is implicit

Coq < Check @nil.
@nil
     : forall A : Set, list A

Coq < Check cons.
cons
     : forall A : Set, A -> list A -> list A
```

Reglas de eliminación

```
Coq < Check list_ind.
list_ind
     : forall (A : Set) (P : list A -> Prop),
       P nil ->
       (forall (a : A) (l : list A), P l -> P (a :: l)) ->
       forall l : list A, P l

Coq < Check list_rec.
list_rec
     : forall (A : Set) (P : list A -> Set),
       P nil ->
       (forall (a : A) (l : list A), P l -> P (a :: l)) ->
       forall l : list A, P l

Coq < Check list_rect.
list_rect
     : forall (A : Set) (P : list A -> Type),
       P nil ->
       (forall (a : A) (l : list A), P l -> P (a :: l)) ->
       forall l : list A, P l
```

Reglas de cómputo

```
Coq < Lemma list_ind_1:forall (A:Set) (P:(list A)->Prop) (vnil:(P nil))
Coq < (vpaso:forall (y:A) (l:(list A)),(P l)->(P (cons y l))),
Coq < (list_ind P vnil vpaso nil)=vnil.
1 subgoal

  ============================
   forall (A : Set) (P : list A -> Prop) (vnil : P nil)
     (vpaso : forall (y : A) (l : list A), P l -> P (y :: l)),
    list_ind P vnil vpaso nil = vnil

list_ind_1 < auto.
```

```
Proof completed.

list_ind_1 < Save.
auto.
list_ind_1 is defined



Coq < Lemma list_ind_2:forall (A:Set) (P:(list A)->Prop) (vnil:(P nil))
Coq < (vpaso:forall (y:A) (l:(list A)),(P l)->(P (cons y l))) (a:A) (L:(list A)),
Coq < (list_ind P vnil vpaso (cons a L))=(vpaso a L (list_ind P vnil vpaso L)).
1 subgoal

  ============================
   forall (A : Set) (P : list A -> Prop) (vnil : P nil)
     (vpaso : forall (y : A) (l : list A), P l -> P (y :: l))
     (a : A) (L : list A),
   list_ind P vnil vpaso (a :: L) = vpaso a L (list_ind P vnil vpaso L)

auto.
Save.

Lemma list_rec_1:(A:Set; P:((list A)->Set);
                  vnil:(P (Nil A));
     vpaso:(y:A; l:(list A))(P l)->(P (Cons A y l)))
       (list_rec A P vnil vpaso (Nil A))=vnil.
Auto.
Save.

Lemma list_rec_1:

Lemma list_rec_2:
```

Pruebas y construcciones

- Probemos que `A/\B -> B/\A`.

```
Theorem com_and: forall A B P : Prop, (A -> B -> P) -> A /\ B -> P.
Proof.
intros A B.
intro H.
apply (and_ind (A:=A) (B:=B) (P:=B /\ A)).
 intros.
   split.
  assumption.
  assumption.
 assumption.
Qed.

Coq < Print com_and.
com_and =
fun (A B : Prop) (H : A /\ B) =>
and_ind (fun (H0 : A) (H1 : B) => conj H1 H0) H
     : forall A B : Prop, A /\ B -> B /\ A
```

- Probemos ahora que `forall (n:nat),0<=n`.