1

Inducción y deducción (2).

Aplicaciones a la definición y verificación de programas.

(Copia de las transparencias)

freire@lfcia.org

2 Tipos Inductivos

- El tipo `A /\ B`

  Reglas de introducción

```
Inductive and [A : Prop; B : Prop]  : Prop
                           :=  conj : A->B->A/\B.

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

  Reglas de eliminación

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

  Reglas de cómputo

```
Coq < Parameters A,B,Q:Prop.
A is assumed
B is assumed
Q is assumed

Coq < Parameter H:A->B->Q.
H is assumed

Coq < Parameters a:A;b:B.
a is assumed
b is assumed

Coq < Eval Simpl in
```

```
      (and_ind A B Q H (conj A B a b)).
     = (H a b)
     : Q
```

- El tipo `nat` *Reglas de introducción*

```
Coq < Inductive nat : Set :=  O : nat | S : nat->nat
Coq < Check nat.
nat: Set
Coq < Check O.
O: nat
Coq < Check S.
S: nat->nat
```

Reglas de eliminación

```
nat_ind: (P:(nat->Prop))(P O)->((n:nat)(P n)->
                                (P (S n)))->(n:nat)(P n)
nat_rec: (P:(nat->Set))(P O)->((n:nat)(P n)->
                                (P (S n)))->(n:nat)(P n)
nat_rect: (P:(nat->Type))(P O)->((n:nat)(P n)->
                                (P (S n)))->(n:nat)(P n)
```

Reglas de cómputo

```
Lemma nat_ind_1:(P:(nat->Prop))
                (init: (P O))
                (paso:(n:nat)(P n)->(P (S n)))
                (nat_ind P init paso O)==init.
Auto.
Save.
Lemma nat_ind_2:(P:(nat->Prop))
                (init: (P O))
                (paso:(n:nat)(P n)->(P (S n)))
                (n:nat)
                   (nat_ind P init paso (S n))
                     ==(paso n (nat_ind P init paso n)).

Auto.
Save.
```

```
 Lemma nat_rec_1:(P:(nat->Set))
                    (init: (P O))
                    (paso:(n:nat)(P n)->(P (S n)))
                    (nat_rec P init paso O)=init.
Auto.
Save.
Lemma nat_rec_2:(P:(nat->Set))
                (init: (P O))
                (paso:(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.
```

3

- **El tipo de los Ordinales (Brouwer)**

  Reglas de introducción

```
Coq < Inductive Ord:Set :=
        Oo:Ord | So:Ord -> Ord | lim:(nat -> Ord)->Ord.

Coq < Check Ord.
Ord
     : Set

Coq < Check So.
So
     : Ord->Ord

Coq < Check lim.
lim
     : (nat->Ord)->Ord
```

  Reglas de eliminación

```
Ord_ind: (P:(Ord->Prop))
        (P Oo)
        ->((o:Ord)(P o)->(P (So o)))
      ->((o:(nat->Ord))((n:nat)(P (o n)))->(P (lim o)))
        ->(o:Ord)(P o)
Ord_rec: (P:(Ord->Set))
        (P Oo)
        ->((o:Ord)(P o)->(P (So o)))
      ->((o:(nat->Ord))((n:nat)(P (o n)))->(P (lim o)))
        ->(o:Ord)(P o)
Ord_rect: (P:(Ord->Type))
        (P Oo)
        ->((o:Ord)(P o)->(P (So o)))
      ->((o:(nat->Ord))((n:nat)(P (o n)))->(P (lim o)))
        ->(o:Ord)(P o)
```

Reglas de cómputo

```
Coq < Variable P:Ord->Set.
P is assumed

Coq < Variable a:(P Oo).
a is assumed

Coq < Variable g:(o:Ord)(P o)->(P (So o)).
g is assumed

Coq < Variable h:(o:(nat->Ord))
            ((n:nat)(P (o n)))->(P (lim o)).
h is assumed

Coq < Variable x:Ord.
x is assumed

Coq < Variable u:(nat -> Ord).
u is assumed

Coq < Eval Simpl in (Ord_rec P a g h Oo).
     = a
```

```
     : (P Oo)
Coq < Eval Simpl in (Ord_rec P a g h (So x)).
     = (g x (Ord_rec P a g h x))
     : (P (So x))
Coq < Eval Simpl in (Ord_rec P a g h (lim u)).
     = (h u [n:nat](Ord_rec P a g h (u n)))
     : (P (lim u))
```

4

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

  Reglas de introducción

```
Coq < Inductive list [A:Set] : Set :=
Coq <  Nil : (list A)
Coq < |Cons : A -> (list A) -> (list A).
Coq < Check list.
list: Set->Set
Coq < Check Nil.
Nil: (A:Set)(list A)
Coq < Check Cons.
Cons: (A:Set)A->(list A)->(list A)
```

  Reglas de eliminación

```
list_ind: (A:Set; P:((list A)->Prop))
        (P (Nil A))
      ->((y:A; l:(list A))(P l)->(P (Cons A y l)))
        ->(l:(list A))(P l)
list_rec: (A:Set; P:((list A)->Set))
        (P (Nil A))
      ->((y:A; l:(list A))(P l)->(P (Cons A y l)))
        ->(l:(list A))(P l)
list_rect: (A:Set; P:((list A)->Type))
        ( (Nil A))
      ->((y:A; l:(list A))(P l)->(P (Cons A y l)))
        ->(l:(list A))(P l)
```

  Reglas de cómputo

```
Lemma list_ind_1:(A:Set; P:((list A)->Prop);
                  vnil:(P (Nil A));
    vpaso:(y:A; l:(list A))(P l)->(P (Cons A y l)))
       (list_ind A P vnil vpaso (Nil A))==vnil.
Auto.
Save.

Lemma list_ind_2:(A:Set; P:((list A)->Prop);
                  vnil:(P (Nil A));
     vpaso:(y:A; l:(list A))(P l)->(P (Cons A y l));
            y:A;l:(list A))
    (list_ind A P vnil vpaso (Cons A y l))
       ==(vpaso y l (list_ind A 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_2:(A:Set; P:((list A)->rec);
                  vnil:(P (Nil A));
     vpaso:(y:A; l:(list A))(P l)->(P (Cons A y l));
            y:A;l:(list A))
    (list_rec A P vnil vpaso (Cons A y l))
           =(vpaso y l (list_rec A P vnil vpaso l)).

Auto.
Save.
```

5 Pruebas y construcciones

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

```
Theorem com_and: A/\B -> B/\A.
```

```
Apply (and_ind A B B/\A).
Intros.
Apply (conj B A); Assumption.
Qed.
```

o equivalentemente:

```
Theorem com_and: A/\B -> B/\A.
Intro H.
Elim H.
Intros.
Split; Assumption.
Qed.

Coq < Print com_and.
com_and = (and_ind A B B/\A [H:A; H0:B]<B,A>{H0,H})
     : A/\B->B/\A
```

donde `<B,A>{H0,H}` es una abreviatura de `(conj A B h0 h1)`:

```
Coq < Check (conj A B h0 h1).
<A,B>{h0,h1}
     : A/\B
```

- Probemos que `(n:nat)(le O n)`.

```
Coq < Lemma uno:(n:nat)(le O n).
1 subgoal

  ============================
   (n:nat)(le O n)

uno < Check (nat_ind [n:nat](le O n)).
(nat_ind [n:nat](le O n))
     : ([n:nat](le O n) O)
 ->((n:nat)([n:nat](le O n) n)->([n0:nat](le O n0) (S n)))
       ->(n:nat)([n0:nat](le O n0) n)

uno < Eval Simpl in ([n:nat](le O n) O)
 ->((n:nat)([n:nat](le O n) n)->([n0:nat](le O n0) (S n)))
```

```
       -
>(n:nat)([n0:nat](le O n0) n).
= (le O O)->((n:nat)(le O n)->(le O (S n)))->(n:nat)(le O n)
     : Prop

uno < Apply (nat_ind [n:nat](le O n)).
2 subgoals

  ============================
   (le O O)

subgoal 2 is:
 (n:nat)(le O n)->(le O (S n))
uno < Constructor 1.
1 subgoal

  ============================
   (n:nat)(le O n)->(le O (S n))

uno < Check le_S.
le_S
     : (n,m:nat)(le n m)->(le n (S m))

uno < Intros.
1 subgoal

  n : nat
  H : (le O n)
  ============================
   (le O (S n))

uno < Constructor 2;Assumption.
Subtree proved!
uno < Restart.

1 subgoal

  ============================
   (n:nat)(le O n)
```

```
uno < Induction n.
2 subgoals

  n : nat
  ============================
   (le O O)

subgoal 2 is:
 (n0:nat)(le O n0)->(le O (S n0))

uno < Undo.
1 subgoal

  ============================
   (n:nat)(le O n)

uno < Intro.
1 subgoal

  n : nat
  ============================
   (le O n)

uno < Elim n.
2 subgoals

  n : nat
  ============================
   (le O O)

subgoal 2 is:
 (n0:nat)(le O n0)->(le O (S n0))
```