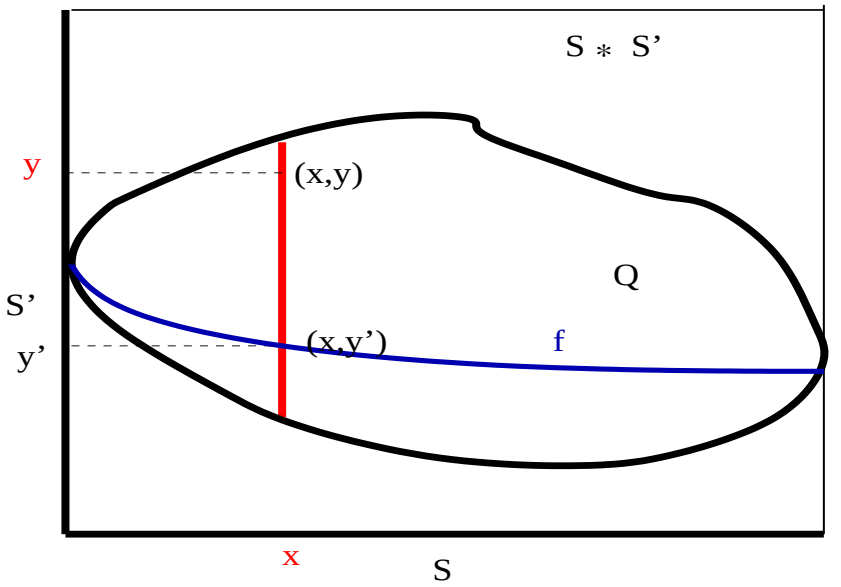
Elección



Si

$$\forall x : S \cdot \exists y : S' \cdot Q(x, y)$$

Entonces

$$\exists f : S \to S' \cdot Q(x, f(x))$$

O más generalmente: Si

$$\forall x : S \cdot P(x) \Rightarrow \exists y : S' \cdot Q(x, y)$$

Entonces

$$\exists f : \{x \in S | P(x)\} \to S' \cdot Q(x, f(x))$$

```
Coq < Section eleccion.

Coq < Variables S,S':Set.
S is assumed
S' is assumed

Coq < Variable Q:S->S'->Prop.
Q is assumed

Coq < Lemma elecc:((x:S){y:S' | (Q x y)})->{f:(S->S') | ((z:S)(Q z (f z)))}.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  ============================
   ((x:S){y:S' | (Q x y)})->{f:(S->S') | ((z:S)(Q z (f z)))}

Coq < Proof.

Coq < Intros.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  H : (x:S){y:S' | (Q x y)}
  ============================
   {f:(S->S') | ((z:S)(Q z (f z)))}

Coq < Split with [s:S](proj1_sig S' [s':S'](Q s s') (H s)).
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  H : (x:S){y:S' | (Q x y)}
  ============================
   (z:S)(Q z (proj1_sig S' [s':S'](Q z s') (H z)))
```

```
Coq < Intro z; Elim (H z); Intros.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  H : (x:S){y:S' | (Q x y)}
  z : S
  x : S'
  p : (Q z x)
  ============================
   (Q z (proj1_sig S' [s':S'](Q z s') (exist S' [y:S'](Q z y) x p)))

Coq < Simpl.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  H : (x:S){y:S' | (Q x y)}
  z : S
  x : S'
  p : (Q z x)
  ============================
   (Q z x)

Coq < Auto.
Subtree proved!

Coq < Defined.
Intros.
Split with [s:S](proj1_sig S' [s':S'](Q s s') (H s)).
Intro z; Elim (H z); Intros.
Simpl.
Auto.
elecc is defined
```

```
Coq < Variable P:S->Prop.
P is assumed

Coq <
Coq < Lemma nelecc:((x:S)(P x)->{y:S' | (Q x y)})->
Coq < {f:({s:S|(P s)}->S') | (H:{s:S|(P s)})let (s,p)=H in (Q s (f H))}.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  P : S->Prop
  ============================
   ((x:S)(P x)->{y:S' | (Q x y)})
   ->{f:({s:S | (P s)}->S') |
        ((H:({s:S | (P s)}))(let (s, _) = H in (Q s (f H))))}

Coq < Proof.

Coq < Intros.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  P : S->Prop
  H : (x:S)(P x)->{y:S' | (Q x y)}
  ============================
   {f:({s:S | (P s)}->S') |
      ((H0:({s:S | (P s)}))(let (s, _) = H0 in (Q s (f H0))))}

Coq < Split
Coq <  with [H0:({s:S | (P s)})]
Coq <        (let (s, p) = H0 in (let (y', _) = (H s p) in y')).
1 subgoal

  S : Set
```

```
  S' : Set
  Q : S->S'->Prop
  P : S->Prop
  H : (x:S)(P x)->{y:S' | (Q x y)}
  ============================
   (H0:({s:S | (P s)}))
    (let (s, _) = H0
     in (Q s (let (s0, p) = H0 in (let (y', _) = (H s0 p) in y'))))
Coq < Intros.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  P : S->Prop
  H : (x:S)(P x)->{y:S' | (Q x y)}
  H0 : {s:S | (P s)}
  ============================
   (let (s, _) = H0
    in (Q s (let (s0, p) = H0 in (let (y', _) = (H s0 p) in y'))))
Coq < Elim H0; Intros.
1 subgoal

  S : Set
  S' : Set
  Q : S->S'->Prop
  P : S->Prop
  H : (x:S)(P x)->{y:S' | (Q x y)}
  H0 : {s:S | (P s)}
  x : S
  p : (P x)
  ============================
   (Q x (let (y', _) = (H x p) in y'))
Coq < Elim (H x p); Intros.
1 subgoal
```

```
  S : Set
  S' : Set
  Q : S->S'->Prop
  P : S->Prop
  H : (x:S)(P x)->{y:S' | (Q x y)}
  H0 : {s:S | (P s)}
  x : S
  p : (P x)
  x0 : S'
  p0 : (Q x x0)
  ============================
   (Q x x0)

Coq < Auto.
Subtree proved!

Coq < Defined.
Intros.
Split
 with [H0:({s:S | (P s)})]
       (let (s, p) = H0 in (let (y', _) = (H s p) in y')).
Intros.
Elim H0; Intros.
Elim (H x p); Intros.
Auto.
nelecc is defined

Coq < End eleccion.
elecc is discharged.
nelecc is discharged.

Coq < Check elecc.
elecc
     : (S,S':Set; Q:(S->S'->Prop))
        ((x:S){y:S' | (Q x y)})->{f:(S->S') | ((z:S)(Q z (f z)))}

Coq < Check nelecc.
nelecc
```

```
     : (S,S':Set; Q:(S->S'->Prop); P:(S->Prop))
        ((x:S)(P x)->{y:S' | (Q x y)})
        ->{f:({s:S | (P s)}->S') |
             ((H:({s:S | (P s)}))(let (s, _) = H in (Q s (f H))))}
```

Por ejemplo, cosideremos el siguiente predicado sobre `nat`:

```
Coq < Inductive Factorial:nat->nat->Prop:=
Coq <         Factorial0: (Factorial O (S O))
Coq <       | FactorialS: (n,p:nat)(Factorial n p)->(Factorial (S n)
Coq <  (mult (S n) p)).
```

---

el programa `elecc` nos permite construir la función factorial:

```
Coq < Check (elecc nat nat [n,m:nat](Factorial n m)).
(elecc nat nat [n,m:nat](Factorial n m))
     : ((x:nat){y:nat | ([n,m:nat](Factorial n m) x y)})
       ->{f:(nat->nat) | ((z:nat)([n,m:nat](Factorial n m) z (f z)))}
```

---

para ello construimos el término `fact`:

```
Coq < Lemma fact:(x:nat){y:nat | ([n,m:nat](Factorial n m) x y)}.
Coq < Proof.
Coq < Induction x.
Coq < Split with (S O); Constructor 1.
Coq <
Coq < Clear x.
Coq < Intros n H; Elim H; Intros t pr.
Coq < Split with (mult (S n) t); Constructor 2;Auto.
Coq < Defined.
```

```
Coq < Check (elecc nat nat [n,m:nat](Factorial n m) fact).
(elecc nat nat [n,m:nat](Factorial n m) fact)
     : {f:(nat->nat) | ((z:nat)([n,m:nat](Factorial n m) z (f z)))}

Coq <
Coq < Definition Fact := let (f,_)=(elecc nat nat [n,m:nat](Factorial n m) fact)
Coq < in f.
Fact is defined

Coq <
Coq < Require Arith.

Coq <
Coq < Eval Compute in (Fact (4)).
     = (24)
     : nat
```

Extracción

Ejemplo:
Especificación de la función de Ackermann

```
Coq < Inductive Ack  : nat->nat->nat->Prop :=
Coq <       AckO : (n:nat)(Ack O n (S n))
Coq <     | AcknO : (n,m:nat)(Ack n (S O) m)->(Ack (S n) O m)
Coq <     | AckSS : (n,m,p,q:nat)
Coq <                 (Ack (S n) m q)->(Ack n q p)->(Ack (S n) (S m) p).
Ack is defined
Ack_ind is defined
```

teorema a demostrar

```
Coq < Theorem ack: (n,m:nat){p:nat | (Ack n m p)}.
1 subgoal

  ============================
   (n,m:nat){p:nat | (Ack n m p)}

Coq < Proof.

Coq < Induction n; Induction m; Clear n; Clear m.
4 subgoals

  ============================
   {p:nat | (Ack (0) (0) p)}
subgoal 2 is:
 (n:nat){p:nat | (Ack (0) n p)}->{p:nat | (Ack (0) (S n) p)}
```


```
subgoal 3 is:
 {p:nat | (Ack (S n0) (0) p)}
subgoal 4 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Split with (S O); Auto.
4 subgoals

  ============================
   (Ack (0) (0) (1))
subgoal 2 is:
 (n:nat){p:nat | (Ack (0) n p)}->{p:nat | (Ack (0) (S n) p)}
subgoal 3 is:
 {p:nat | (Ack (S n0) (0) p)}
subgoal 4 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Constructor 1.
3 subgoals

  ============================
   (n:nat){p:nat | (Ack (0) n p)}->{p:nat | (Ack (0) (S n) p)}
subgoal 2 is:
 {p:nat | (Ack (S n0) (0) p)}
subgoal 3 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Intros.
3 subgoals

  n : nat
  H : {p:nat | (Ack (0) n p)}
  ============================
   {p:nat | (Ack (0) (S n) p)}
subgoal 2 is:
 {p:nat | (Ack (S n0) (0) p)}
subgoal 3 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
```

```
Coq < Split with (S (S n)).
3 subgoals

  n : nat
  H : {p:nat | (Ack (0) n p)}
  ============================
   (Ack (0) (S n) (S (S n)))
subgoal 2 is:
 {p:nat | (Ack (S n0) (0) p)}
subgoal 3 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Apply (AckO (S n)).
2 subgoals

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  ============================
   {p:nat | (Ack (S n0) (0) p)}
subgoal 2 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Case (H (S O)).
2 subgoals

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  ============================
   (x:nat)(Ack n0 (1) x)->{p:nat | (Ack (S n0) (0) p)}
subgoal 2 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Intros.
2 subgoals

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  x : nat
```

```
  a : (Ack n0 (1) x)
  ============================
   {p:nat | (Ack (S n0) (0) p)}
subgoal 2 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Split with x.
2 subgoals

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  x : nat
  a : (Ack n0 (1) x)
  ============================
   (Ack (S n0) (0) x)
subgoal 2 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Auto.
2 subgoals

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  x : nat
  a : (Ack n0 (1) x)
  ============================
   (Ack (S n0) (0) x)
subgoal 2 is:
 (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Constructor 2; Auto.
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  ============================
   (n:nat){p:nat | (Ack (S n0) n p)}->{p:nat | (Ack (S n0) (S n) p)}
Coq < Intros.
```








```
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  ============================
   {p:nat | (Ack (S n0) (S n) p)}

Coq < Case H0.
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  ============================
   (x:nat)(Ack (S n0) n x)->{p:nat | (Ack (S n0) (S n) p)}

Coq < Intros.
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  x : nat
  a : (Ack (S n0) n x)
  ============================
   {p:nat | (Ack (S n0) (S n) p)}

Coq < Case (H x).
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
```

```
  x : nat
  a : (Ack (S n0) n x)
  ============================
   (x0:nat)(Ack n0 x x0)->{p:nat | (Ack (S n0) (S n) p)}

Coq < Intros.
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  x : nat
  a : (Ack (S n0) n x)
  x0 : nat
  a0 : (Ack n0 x x0)
  ============================
   {p:nat | (Ack (S n0) (S n) p)}

Coq < Split with x0.
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  x : nat
  a : (Ack (S n0) n x)
  x0 : nat
  a0 : (Ack n0 x x0)
  ============================
   (Ack (S n0) (S n) x0)

Coq < Apply (AckSS n0 n x0 x).
2 subgoals

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
```








```
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  x : nat
  a : (Ack (S n0) n x)
  x0 : nat
  a0 : (Ack n0 x x0)
  ============================
   (Ack (S n0) n x)
subgoal 2 is:
 (Ack n0 x x0)
Coq < Auto.
1 subgoal

  n0 : nat
  H : (m:nat){p:nat | (Ack n0 m p)}
  n : nat
  H0 : {p:nat | (Ack (S n0) n p)}
  x : nat
  a : (Ack (S n0) n x)
  x0 : nat
  a0 : (Ack n0 x x0)
  ============================
   (Ack n0 x x0)
Coq < Auto.
Subtree proved!
Coq < Defined.
Induction n; Induction m; Clear n; Clear m.
Split with (S O); Auto.
Constructor 1.
Intros.
Split with (S (S n)).
Apply (AckO (S n)).
Case (H (S O)).
Intros.
Split with x.
```

```
Auto.
Constructor 2; Auto.
Intros.
Case H0.
Intros.
Case (H x).
Intros.
Split with x0.
Apply (AckSS n0 n x0 x).
Auto.
Auto.
ack is defined
```

Algunos cálculos

```
Coq < Require Arith.

Coq <
Coq < Definition pr1:=[A:Set;P:A->Prop][H:{x:A|(P x)}]
Coq <              let (x,p)=H in x.
pr1 is defined

Coq <
Coq < Eval Compute in (ack (1) (0)).
     = (exist nat [p:nat](Ack (1) (0) p) (2) (AcknO (0) (2) (AckO (1))))
     : {p:nat | (Ack (1) (0) p)}

Coq <
Coq < Eval Compute in (pr1 nat ? (ack (1) (0))).
     = (2)
     : nat

Coq <
Coq < Eval Compute in (ack (2) (1)).
     = (exist nat [p:nat](Ack (2) (1) p) (5)
         (AckSS (1) (0) (5) (3)
           (AcknO (1) (3)
             (AckSS (0) (0) (3) (2) (AcknO (0) (2) (AckO (1)))
               (AckO (2))))
           (AckSS (0) (2) (5) (4)
             (AckSS (0) (1) (4) (3)
               (AckSS (0) (0) (3) (2) (AcknO (0) (2) (AckO (1)))
                 (AckO (2))) (AckO (3))) (AckO (4)))))
     : {p:nat | (Ack (2) (1) p)}

Coq <
```

```
Coq < Eval Compute in (pr1 ? ? (ack (2) (1))).
     = (5)
     : nat
Coq <
```







El término ack y la extracción de su parte constructiva.

λ–término tipado ack

```
ack =
[n:nat]
 (nat_rec [n0:nat](m:nat)p:nat | (Ack n0 m p)
   [m:nat]
    (nat_rec [n0:nat]p:nat | (Ack O n0 p)
      (exist nat [p:nat](Ack O O p) (S O) (AckO O))
      [n0:nat; _:(p:nat | (Ack O n0 p))]
       (exist nat [p:nat](Ack O (S n0) p) (S (S n0)) (AckO (S n0))) m)
   [n0:nat; H:((m:nat)p:nat | (Ack n0 m p)); m:nat]
    (nat_rec [n1:nat]p:nat | (Ack (S n0) n1 p)
      (let (x, a) = (H (S O))
       in (exist nat [p:nat](Ack (S n0) O p) x (AcknO n0 x a)))
      [n1:nat; H0:(p:nat | (Ack (S n0) n1 p))]
       (let (x, a) = H0
        in (let (x0, a0) = (H x)
            in (exist nat [p:nat](Ack (S n0) (S n1) p) x0
                 (AckSS n0 n1 x0 x a a0)))) m) n)
     : (n,m:nat)p:nat | (Ack n m p)
```

la extracción a CAML del contenido computacional de ack

```
Coq < Recursive Extraction ack.
type nat =
    O
  | S of nat

let nat_rec f0 f =
  let rec f1 = function
    O -> f0
```

```
  | S n0 -> f n0 (f1 n0)
  in f1

let ack n =
  nat_rec (fun m -> nat_rec (S O) (fun n0 h -> S (S n0)) m)
    (fun n0 h m -> nat_rec (h (S O)) (fun n1 h0 -> h h0) m) n
```

en CAML

```
freire@emcpc:~$ ocaml
        Objective Caml version 3.01

# type nat =
    O
  | S of nat    ;;
type nat = O | S of nat
# let nat_rec f0 f =
  let rec f1 = function
    O -> f0
  | S n0 -> f n0 (f1 n0)
  in f1        ;;
val nat_rec : 'a -> (nat -> 'a -> 'a) -> nat -> 'a = <fun>
# let ack n =
  nat_rec (fun m -> nat_rec (S O) (fun n0 h -> S (S n0)) m)
    (fun n0 h m -> nat_rec (h (S O)) (fun n1 h0 -> h h0) m) n    ;;
val ack : nat -> nat -> nat = <fun>
# let rec natint = function
    O -> 0
  |S x -> (natint x)+1;;
val natint : nat -> int = <fun>
# let rec intnat = function
   0 -> O
  |x -> (S (intnat (x-1)));;
val intnat : int -> nat = <fun>
# natint ( ack (intnat 3) (intnat 2));;
- : int = 29
```

Prueba automática en Coq

Algunos mecanismos de decisión en Coq.

- cálculo proposicional intuicionista (Tésis de Master de César Muñoz (1994)).

```
Goal (x:nat)(P:nat -> Prop)(x=O  (P x)) -> (~x=O) -> (P x).
1 subgoal
  ============================
   (x:nat; P,Q:(nat->Prop))(P x)(Q x)->~(P x)->(Q x)
Unnamed_thm < Tauto.
Subtree proved!
```

- Aritmética de Presburger:

```
freire@emcpc:~$ coqtop
Welcome to Coq 7.0 (April 2001)
Coq < Require Omega.
Coq < Goal (m,n:Z)~‘1+2*m=2*n‘.
1 subgoal
  ============================
   (m,n:Z)‘1+2*m <> 2*n‘
Unnamed_thm < Intros.
1 subgoal
  m : Z
  n : Z
  ============================
   ‘1+2*m <> 2*n‘
Unnamed_thm < Omega.
Subtree proved!
```