1 Tipos Inductivos (Cont.)

- **El tipo** `False`

```
Coq < Print False.
Inductive False  : Prop :=

Coq < Print False_ind.
False_ind = [P:Prop; f:False]<P>Cases f of end
     : (P:Prop)False->P

Coq < Print False_rec.
False_rec =
[P:Set; H:False]
 [H0:=(False_ind true=false H)]
  <[b:bool](Q P b)>Cases H0 of refl_equal => tt end
     : (P:Set)False->P
```

Las tácticas `Absurd` *proposicion* y `Contradiction`.

```
Coq <
Coq < Require Arith.

Coq <
Coq < Lemma contr: (p:False;P:Prop)P.
1 subgoal

  ============================
   False->(P:Prop)P

Coq < Proof.

Coq < Intros.
1 subgoal

  p : False
  P : Prop
  ============================
   P

Coq < Contradiction.
Subtree proved!

Coq < Qed.
Intros.
```

```
Contradiction.
contr is defined

Coq <
Coq < Lemma abs:(P:Prop)(1)=0->P.
1 subgoal

  ============================
   (P:Prop)(1)=(0)->P

Coq < Proof.

Coq < Intros.
1 subgoal

  P : Prop
  H : (1)=(0)
  ============================
   P

Coq < Absurd (1)=0;Auto.
Subtree proved!

Coq < Qed.
Intros.
Absurd (1)=0; Auto.
abs is defined

Coq <
```

2

- **El tipo igualdad**

```
Coq < Inductive Eq [A:Set;x:A]:A -> Prop := Refl_equal : (Eq A x x).

Coq < Check Eq.
Eq: (A:Set)A->A->Prop
Coq < Check Refl_equal.
Refl_equal: (A:Set; x:A)(Eq A x x)
Eq_ind: (A:Set; x:A; P:(A->Prop))(P x)->(y:A)(Eq A x y)->(P y)
Eq_rec: (A:Set; x:A; P:(A->Set))(P x)->(y:A)(Eq A x y)->(P y)
```

El definido en el propio sistema `(eq A x y)` se abrevia `x=y`.

```
Coq < Check eq.
eq: (A:Set)A->A->Prop
Coq < Check eq_ind.
eq_ind
     : (A:Set; x:A; P:(A->Prop))(P x)->(y:A)x=y->(P y)
```

La táctica `Rewrite`:

```
Coq < Require ZArith.
Coq <

Coq < Check Zmult_plus_distr.
Zmult_plus_distr
     : (n,m,p:Z)`(n+m)*p = n*p+m*p`

Coq < Check Zmult_1_n.
Zmult_1_n
     : (n:Z)`1*n = n`

Coq < Lemma ejemplo1:(n,x:Z)`n*x+x=(n+1)*x`.
1 subgoal

  ============================
   (n,x:Z)`n*x+x = (n+1)*x`

Coq < Proof.

Coq < Intros n x.
1 subgoal

  n : Z
  x : Z
  ============================
   `n*x+x = (n+1)*x`

Coq < Rewrite Zmult_plus_distr.
1 subgoal

  n : Z
  x : Z
  ============================
   `n*x+x = n*x+1*x`

Coq < Rewrite Zmult_1_n.
1 subgoal

  n : Z
```

```
  x : Z
  ============================
   `n*x+x = n*x+x`

Coq < Trivial.
Subtree proved!

Coq < Qed.
Intros n x.
Rewrite Zmult_plus_distr.
Rewrite Zmult_1_n.
Trivial.
ejemplo1 is defined
```

Se puede usar con la simetría de la igualdad: `Rewrite <-` e.

Incluso reemplazar (`Replace T with U`) un tipo con otro que habrá que probar que es igual a él.

- La táctica `discriminate` y el carácter inyectivo (`Injection`) de los constructores de un tipo

---

```
Coq < Lemma ejemplo2:(n:nat)~(S n)=0.
1 subgoal

  ============================
   (n:nat)~(S n)=(0)

Coq < Proof.

Coq < Intros; Discriminate.
Subtree proved!

Coq < Qed.
Intros; Discriminate.
ejemplo2 is defined

Coq <
Coq < Inductive T [A:Set]:Set:=
Coq <   c:A->(T A).
T is defined
T_ind is defined
T_rec is defined
T_rect is defined

Coq <
Coq < Lemma ejemplo3:(A:Set;x,y:A)((c A x)=(c A y))-> (x=y).
```

```
1 subgoal

  ============================
   (A:Set; x,y:A)(c A x)=(c A y)->x=y

Coq < Proof.

Coq < Intros.
1 subgoal

  A : Set
  x : A
  y : A
  H : (c A x)=(c A y)
  ============================
   x=y

Coq < Injection H.
1 subgoal

  A : Set
  x : A
  y : A
  H : (c A x)=(c A y)
  ============================
   x=y->x=y

Coq < Auto.
Subtree proved!

Coq < Qed.
Intros.
Injection H.
Auto.
ejemplo3 is defined
```

Nótese que:

```
 Eval Compute in ([v:(T A)]Cases v of (c a) => a end (c A x)).
     = x
     : A
```

Por lo tanto:

```
Coq <
```

```
Coq <
Coq < Lemma ejemplo4:(A:Set;x,y:A)((c A x)=(c A y))-> (x=y).
1 subgoal

  ============================
   (A:Set; x,y:A)(c A x)=(c A y)->x=y

Coq < Proof.

Coq < Intros A x y H.
1 subgoal

  A : Set
  x : A
  y : A
  H : (c A x)=(c A y)
  ============================
   x=y

Coq < Change ([v:(T A)]Cases v of (c a) => a end (c A x))=
Coq < ([v:(T A)]Cases v of (c a) => a end (c A y)).
1 subgoal

  A : Set
  x : A
  y : A
  H : (c A x)=(c A y)
  ============================
   ([v:(T A)]Cases v of (c a) => a end (c A x))
    =([v:(T A)]Cases v of (c a) => a end (c A y))

Coq < Rewrite H.
1 subgoal

  A : Set
  x : A
  y : A
  H : (c A x)=(c A y)
  ============================
   y=y

Coq < Auto.
Subtree proved!

Coq < Qed.
Intros A x y H.
Change
  ([v:(T A)]Cases (v) of (c a) => a end (c A x))
```

```
   =([v:(T A)]Cases (v) of (c a) => a end (c A y)) .
Rewrite H.
Auto.
ejemplo4 is defined
Coq <
```

3

- **El tipo existencial** En lógica constructiva, una prueba de la proposición

$$\exists x : A \cdot P(x)$$

consiste en un par (a, p) donde a es de type A y p es una prueba de $(P\ a)$. Por ello, aparece como una generalización del producto cartesiano de conjuntos donde el segundo elemento del par depende del primero.

```
Coq < Inductive ex [A:Set; P:A->Prop] : Prop :=
      ex_intro : (x:A)(P x)->(ex A P)
Coq < Check ex.
ex: (A:Set)(A->Prop)->Prop
Coq < Check ex_intro.
ex_intro: (A:Set; P:(A->Prop); x:A)(P x)->(Ex P)

Coq <  Check [A:Set;P:A->Prop;x:A;p:(P x)](ex_intro A P x p).
[A:Set; P:(A->Prop); x:A; p:(P x)](ex_intro A P x p)
     : (A:Set; P:(A->Prop); x:A)(P x)->(Ex P)

Coq <  Check [A:Set;P:A->Prop;x:A;p:(P x)](ex_intro A [a:A](P a) x p).
[A:Set; P:(A->Prop); x:A; p:(P x)](ex_intro A [a:A](P a) x p)
     : (A:Set; P:(A->Prop); x:A)(P x)->(EX a:A | (P a))
```

Nótese que Coq simplifica `(ex A P)` a `(Ex P)` si el tipo A se conoce por el contexto.

```
Coq < Print Ex.
 Syntax Macro Ex = (ex ?)
ex_ind: (A:Set; P:(A->Prop); P0:Prop)((x:A)(P x)->P0)->(ex A P)->P0
```

- **La versión constructiva del tipo existencial** Dados $A : Set$ y un predicado $P : A \rightarrow Prop$, se define el tipo

$$\{x \in A | (P\ x)\} : \texttt{Set}$$

cuyos *elementos* son los pares (x, p) donde, como el caso anterior, $x : A$ representa el *testigo*, y $p : (P\ x)$ es la justificación de que x *satisface* el predicado P.

```
Coq < Inductive sig [A:Set; P:A->Prop] : Set :=
Coq <       exist : (x:A)(P x)->(sig A P).
Coq < sig is defined
sig_ind is defined
sig_rec is defined
sig_rect is defined

Coq < Check sig.
sig
     : (A:Set)(A->Prop)->Set

Coq < Check (A:Set;P:A->Prop)(sig A [x:A](P x)).
(A:Set; P:(A->Prop))(sig A [x:A](P x))
     : Set

Coq < Check (A:Set;P:A->Prop)(sig A P).
(A:Set; P:(A->Prop))(sig A P)
     : Set

Coq < Check exist.
exist
     : (A:Set; P:(A->Prop); x:A)(P x)->(sig A P)

Coq < Check [A:Set;P:A->Prop;x:A;p:(P x)](exist A P x p).
[A:Set; P:(A->Prop); x:A; p:(P x)](exist A P x p)
     : (A:Set; P:(A->Prop); x:A)(P x)->(sig A P)

Coq < Check [A:Set;P:A->Prop;x:A;p:(P x)](exist A [a:A](P a) x p).
[A:Set; P:(A->Prop); x:A; p:(P x)](exist A [a:A](P a) x p)
     : (A:Set; P:(A->Prop); x:A)(P x)->(sig A [a:A](P a))

Coq <
Coq < Check sig_ind.
sig_ind
     : (A:Set; P:(A->Prop); P0:((sig A P)->Prop))
        ((x:A; p:(P x))(P0 (exist A P x p)))->(s:(sig A P))(P0 s)

Coq < Check sig_rec.
sig_rec
     : (A:Set; P:(A->Prop); P0:((sig A P)->Set))
        ((x:A; p:(P x))(P0 (exist A P x p)))->(s:(sig A P))(P0 s)

Coq < Check sig_rect.
sig_rect
     : (A:Set; P:(A->Prop); P0:((sig A P)->Type))
```

```
        ((x:A; p:(P x))(P0 (exist A P x p)))->(s:(sig A P))(P0 s)
Coq <
Coq < Definition pr1:=[A:Set;P:A->Prop;H:{x:A | (P x)}] let (a,p)=H in a.
pr1 is defined
Coq < Check pr1.
pr1
     : (A:Set; P:(A->Prop)){x:A | (P x)}->A
Coq < Definition pr2:=[A:Set;P:A->Prop;H:{x:A | (P x)}]
Coq <        <[H:{x:A | (P x)}](P (pr1 A P H))> let (a,p)=H in p.
pr2 is defined
Coq < Check pr2.
pr2
     : (A:Set; P:(A->Prop); H:({x:A | (P x)}))(P (pr1 A P H))
```

4

- **La relación expresando que un numero r es la suma de m y otro n**

```
Coq < Inductive Plus [n:nat]:nat -> nat->Prop :=
Coq < Plus0 : (Plus n O n)
Coq < |PlusS : (m,r:nat)(Plus n m r) -> (Plus n (S m) (S r)).
Coq < Check Plus.
Plus: nat->nat->nat->Prop
Coq < Check Plus0.
Plus0: (n:nat)(Plus n O n)
Coq < Check PlusS.
PlusS: (n,m,r:nat)(Plus n m r)->(Plus n (S m) (S r))
```

5

Definiciones mutuamente inductivas.

- **Paridad**

```
Coq < Mutual Inductive
Coq <   Even : nat -> Prop  :=
Coq <     even0 : (Even O)
Coq <   | evenS : (n:nat)(Odd n) -> (Even (S n))
Coq < with
```

```
Coq <   Odd  : nat -> Prop :=
Coq <     odd1  : (Odd (S O))
Coq <   | oddS  : (n:nat)(Even n) -> (Odd (S n)).
Even_ind: (P:(nat->Prop))
        (P O)->((n:nat)(Odd n)->(P (S n)))->(n:nat)(Even n)->(P n)
Odd_ind: (P:(nat->Prop))
        (P (S O))->((n:nat)(Even n)->(P (S n)))->(n:nat)(Odd n)->(P n)
```

- **Árboles y bosques**

```
Coq < Mutual Inductive tree:Set := node:forest -> tree
Coq < with forest:Set := emptyf:forest
Coq <                  | consf:tree -> forest -> forest.
Coq < Check tree.
tree: Set
Coq < Check forest.
forest: Set
Coq < Check node.
node: forest->tree
Coq < Check emptyf.
emptyf: forest
Coq < Check consf.
consf: tree->forest->forest
```

6

División Euclídea.

```
Coq < Require Export Arith.

Coq <
Coq < Lemma Lt_O_Sn:(n:nat)(lt O (S n)).
1 subgoal

  ============================
   (n:nat)(lt (0) (S n))

Coq < Proof.

Coq < Induction n.
2 subgoals

  n : nat
  ============================
   (lt (0) (1))
```

```
subgoal 2 is:
 (n0:nat)(lt (0) (S n0))->(lt (0) (S (S n0)))
```

```
Coq < Red.
2 subgoals

  n : nat
  ============================
   (le (1) (1))
subgoal 2 is:
 (n0:nat)(lt (0) (S n0))->(lt (0) (S (S n0)))
```

```
Coq < Constructor 1.
1 subgoal

  n : nat
  ============================
   (n0:nat)(lt (0) (S n0))->(lt (0) (S (S n0)))
```

```
Coq < Intros.
1 subgoal

  n : nat
  n0 : nat
  H : (lt (0) (S n0))
  ============================
   (lt (0) (S (S n0)))
```

```
Coq < Red in H.
1 subgoal

  n : nat
  n0 : nat
  H : (le (1) (S n0))
  ============================
   (lt (0) (S (S n0)))
```

```
Coq < Red.
1 subgoal

  n : nat
  n0 : nat
  H : (le (1) (S n0))
  ============================
   (le (1) (S (S n0)))
```

```
Coq < Constructor 2.
1 subgoal
```

```
  n : nat
  n0 : nat
  H : (le (1) (S n0))
  ============================
   (le (1) (S n0))

Coq < Assumption.
Subtree proved!

Coq < Defined.
Induction n.
Red.
Constructor 1.
Intros.
Red in H.
Red.
Constructor 2.
Assumption.
Lt_O_Sn is defined

Coq <
Coq < Lemma Lt_eq_lt_dec:(n,m:nat){(lt n m)}+{n=m}+{(lt m n)}.
1 subgoal

  ============================
   (n,m:nat){(lt n m)}+{n=m}+{(lt m n)}

Coq < Proof.

Coq < Induction n; Induction m; Auto with arith.
1 subgoal

  n : nat
  n0 : nat
  H : (m:nat){(lt n0 m)}+{n0=m}+{(lt m n0)}
  m : nat
  ============================
   (n1:nat)
    {(lt (S n0) n1)}+{(S n0)=n1}+{(lt n1 (S n0))}
    ->{(lt (S n0) (S n1))}+{(S n0)=(S n1)}+{(lt (S n1) (S n0))}

Coq < Intros q H'; Elim (H q).
2 subgoals

  n : nat
  n0 : nat
  H : (m:nat){(lt n0 m)}+{n0=m}+{(lt m n0)}
```

```
  m : nat
  q : nat
  H' : {(lt (S n0) q)}+{(S n0)=q}+{(lt q (S n0))}
  ============================
   {(lt n0 q)}+{n0=q}
   ->{(lt (S n0) (S q))}+{(S n0)=(S q)}+{(lt (S q) (S n0))}
subgoal 2 is:
 (lt q n0)->{(lt (S n0) (S q))}+{(S n0)=(S q)}+{(lt (S q) (S n0))}
```

```
Coq < Induction 1; Auto with arith.
1 subgoal
```

```
  n : nat
  n0 : nat
  H : (m:nat){(lt n0 m)}+{n0=m}+{(lt m n0)}
  m : nat
  q : nat
  H' : {(lt (S n0) q)}+{(S n0)=q}+{(lt q (S n0))}
  ============================
   (lt q n0)->{(lt (S n0) (S q))}+{(S n0)=(S q)}+{(lt (S q) (S n0))}
```

```
Coq < Auto with arith.
Subtree proved!
```

```
Coq < Defined.
Induction n; Induction m; Auto with arith.
Intros q H'; Elim (H q).
Induction 1; Auto with arith.
Auto with arith.
Lt_eq_lt_dec is defined
```

```
Coq <
Coq < Theorem Lt_n_S : (n,m:nat)(lt n m)->(lt (S n) (S m)).
1 subgoal
```

```
  ============================
   (n,m:nat)(lt n m)->(lt (S n) (S m))
```

```
Coq < Proof.
```

```
Coq < Auto with arith.
Subtree proved!
```

```
Coq < Defined.
Auto with arith.
Lt_n_S is defined
```

```
Coq <
Coq < Theorem Lt_S_n : (n,m:nat)(lt (S n) (S m))->(lt n m).
```

```
1 subgoal

  ============================
   (n,m:nat)(lt (S n) (S m))->(lt n m)

Coq < Proof.

Coq < Auto with arith.
Subtree proved!

Coq < Defined.
Auto with arith.
Lt_S_n is defined

Coq <
Coq < Hints Resolve Lt_O_Sn Lt_eq_lt_dec Lt_n_S Lt_S_n.

Coq <
Coq < Lemma  div_euclid: (a,b:nat)
Coq <  {q:(nat*nat) |(a=(plus (Snd q) (mult (Fst q) (S b)))
Coq <                                  /\(lt (Snd q) (S b)))}.
1 subgoal

  ============================
   (a,b:nat)
    {q:(nat*nat) |
       (a=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Proof.

Coq < Intro a.
1 subgoal

  a : nat
  ============================
   (b:nat)
    {q:(nat*nat) |
       (a=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Elim a.
2 subgoals

  a : nat
  ============================
   (b:nat)
    {q:(nat*nat) |
       ((0)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 (n:nat)
```

```
  ((b:nat)
    {q:(nat*nat) |
       (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
  ->(b:nat)
     {q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Intros.
2 subgoals

  a : nat
  b : nat
  ============================
   {q:(nat*nat) |
      ((0)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 (n:nat)
  ((b:nat)
    {q:(nat*nat) |
       (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
  ->(b:nat)
     {q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Split with (0,0).
2 subgoals

  a : nat
  b : nat
  ============================
   (0)=(plus (Snd ((0),(0))) (mult (Fst ((0),(0))) (S b)))
   /\(lt (Snd ((0),(0))) (S b))
subgoal 2 is:
 (n:nat)
  ((b:nat)
    {q:(nat*nat) |
       (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
  ->(b:nat)
     {q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Simpl.
2 subgoals

  a : nat
  b : nat
  ============================
```

```
   (0)=(0)/\(lt (0) (S b))
subgoal 2 is:
 (n:nat)
  ((b:nat)
    {q:(nat*nat) |
       (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
  ->(b:nat)
     {q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Split.
3 subgoals

  a : nat
  b : nat
  ============================
   (0)=(0)
subgoal 2 is:
 (lt (0) (S b))
subgoal 3 is:
 (n:nat)
  ((b:nat)
    {q:(nat*nat) |
       (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
  ->(b:nat)
     {q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Auto.
2 subgoals

  a : nat
  b : nat
  ============================
   (lt (0) (S b))
subgoal 2 is:
 (n:nat)
  ((b:nat)
    {q:(nat*nat) |
       (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
  ->(b:nat)
     {q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq <
Coq < Apply Lt_O_Sn.
1 subgoal
```

```
  a : nat
  ============================
   (n:nat)
    ((b:nat)
      {q:(nat*nat) |
         (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))})
    ->(b:nat)
       {q:(nat*nat) |
          ((S n)=(plus (Snd q) (mult (Fst q) (S b)))
           /\(lt (Snd q) (S b)))}
Coq <
Coq < Intros.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Elim H with b.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  ============================
   (x:(nat*nat))
    n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
    ->{q:(nat*nat) |
         ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Intros.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
```

```
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Elim Lt_eq_lt_dec with (S (Snd x)) (S b).
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  ============================
   {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
   ->{q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Intro.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 (lt (S b) (S (Snd x)))
```

```
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Elim a0; Intros.
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  a1 : (lt (S (Snd x)) (S b))
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 {q:(nat*nat) |
    ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 3 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Split with ((Fst x),(S (Snd x))).
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  a1 : (lt (S (Snd x)) (S b))
  ============================
   (S n)
    =(plus (Snd ((Fst x),(S (Snd x))))
       (mult (Fst ((Fst x),(S (Snd x)))) (S b)))
   /\(lt (Snd ((Fst x),(S (Snd x)))) (S b))
subgoal 2 is:
```

```
 {q:(nat*nat) |
    ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 3 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Simpl.
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  a1 : (lt (S (Snd x)) (S b))
  ============================
   (S n)=(S (plus (Snd x) (mult (Fst x) (S b))))
   /\(lt (S (Snd x)) (S b))
subgoal 2 is:
 {q:(nat*nat) |
    ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 3 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Split; Trivial.
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  a1 : (lt (S (Snd x)) (S b))
  ============================
   (S n)=(S (plus (Snd x) (mult (Fst x) (S b))))
```

```
subgoal 2 is:
 {q:(nat*nat) |
    ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 3 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Elim p; Intros.
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  a1 : (lt (S (Snd x)) (S b))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   (S n)=(S (plus (Snd x) (mult (Fst x) (S b))))
subgoal 2 is:
 {q:(nat*nat) |
    ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 3 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Auto with arith.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
```

```
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq <
Coq < Intros.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Split with ((S (Fst x)),O).
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  ============================
   (S n)
    =(plus (Snd ((S (Fst x)),(0))) (mult (Fst ((S (Fst x)),(0))) (S b)))
```

```
   /\(lt (Snd ((S (Fst x)),(0))) (S b))
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Simpl.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  ============================
   (S n)=(S (plus b (mult (Fst x) (S b))))/\(lt (0) (S b))
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Split; Auto with arith.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  ============================
   (S n)=(S (plus b (mult (Fst x) (S b))))
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Rewrite <- b0.
```

```
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  ============================
   (S n)=(S (plus b (mult (Fst x) (S (Snd x)))))
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Elim p; Intros.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   (S n)=(S (plus b (mult (Fst x) (S (Snd x)))))
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Rewrite H0.
2 subgoals

  a : nat
  n : nat
```

```
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  a0 : {(lt (S (Snd x)) (S b))}+{(S (Snd x))=(S b)}
  b0 : (S (Snd x))=(S b)
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   (S (plus (Snd x) (mult (Fst x) (S b))))
    =(S (plus b (mult (Fst x) (S (Snd x)))))
subgoal 2 is:
 (lt (S b) (S (Snd x)))
 ->{q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq < Auto with arith.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  ============================
   (lt (S b) (S (Snd x)))
   ->{q:(nat*nat) |
        ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
Coq <
Coq < Intros.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
```

```
  b0 : (lt (S b) (S (Snd x)))
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Elim p; Intros.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   {q:(nat*nat) |
      ((S n)=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}

Coq < Absurd (lt b (Snd x)).
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   ~(lt b (Snd x))
subgoal 2 is:
 (lt b (Snd x))

Coq < Auto with arith.
2 subgoals

  a : nat
  n : nat
```

```
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   ~(lt b (Snd x))
subgoal 2 is:
 (lt b (Snd x))
```

```
Coq < Red; Intros.
2 subgoals
```

```
  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  H2 : (lt b (Snd x))
  ============================
   False
subgoal 2 is:
 (lt b (Snd x))
```

```
Coq < Red in H1.
2 subgoals
```

```
  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
```

```
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (lt b (Snd x))
  ============================
   False
subgoal 2 is:
 (lt b (Snd x))
```

```
Coq < Red in H2.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
      {q:(nat*nat) |
         (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   False
subgoal 2 is:
 (lt b (Snd x))
```

```
Coq < Cut (le (S (Snd x)) (Snd x)).
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
      {q:(nat*nat) |
         (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   (le (S (Snd x)) (Snd x))->False
subgoal 2 is:
```

```
 (le (S (Snd x)) (Snd x))
subgoal 3 is:
 (lt b (Snd x))
Coq < Fold .
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   (le (S (Snd x)) (Snd x))->False
subgoal 2 is:
 (le (S (Snd x)) (Snd x))
subgoal 3 is:
 (lt b (Snd x))
Coq < Change ~(le (S (Snd x)) (Snd x)) .
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   ~(le (S (Snd x)) (Snd x))
subgoal 2 is:
 (le (S (Snd x)) (Snd x))
subgoal 3 is:
```

```
 (lt b (Snd x))

Coq < Apply le_Sn_n.
2 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   (le (S (Snd x)) (Snd x))
subgoal 2 is:
 (lt b (Snd x))

Coq < Apply (le_trans (S (Snd x)) (S b) (Snd x)).
3 subgoals

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   (le (S (Snd x)) (S b))
subgoal 2 is:
 (le (S b) (Snd x))
subgoal 3 is:
 (lt b (Snd x))

Coq < Assumption.
2 subgoals
```

```
  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (le (S (Snd x)) (S b))
  H2 : (le (S b) (Snd x))
  ============================
   (le (S b) (Snd x))
subgoal 2 is:
 (lt b (Snd x))

Coq <
Coq < Assumption.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   (lt b (Snd x))

Coq <
Coq < Apply Lt_S_n.
1 subgoal

  a : nat
  n : nat
  H : (b:nat)
       {q:(nat*nat) |
          (n=(plus (Snd q) (mult (Fst q) (S b)))/\(lt (Snd q) (S b)))}
  b : nat
  x : nat*nat
```

```
  p : n=(plus (Snd x) (mult (Fst x) (S b)))/\(lt (Snd x) (S b))
  b0 : (lt (S b) (S (Snd x)))
  H0 : n=(plus (Snd x) (mult (Fst x) (S b)))
  H1 : (lt (Snd x) (S b))
  ============================
   (lt (S b) (S (Snd x)))

Coq < Assumption.
Subtree proved!

Coq < Defined.
Intro a.
Elim a.
Intros.
Split with (O,O).
Simpl.
Split.
Auto.
Apply Lt_O_Sn.
Intros.
Elim H with b.
Intros.
Elim Lt_eq_lt_dec with (S (Snd x)) (S b).
Intro.
Elim a0; Intros.
Split with ((Fst x),(S (Snd x))).
Simpl.
Split; Trivial.
Elim p; Intros.
Auto with arith.
Intros.
Split with ((S (Fst x)),O).
Simpl.
Split; Auto with arith.
Rewrite <- b0.
Elim p; Intros.
Rewrite H0.
Auto with arith.
Intros.
Elim p; Intros.
Absurd (lt b (Snd x)).
Auto with arith.
Red; Intros.
Red in H1.
Red in H2.
Cut (le (S (Snd x)) (Snd x)).
```

```
Fold .
Change ~(le (S (Snd x)) (Snd x)) .
Apply le_Sn_n.
Apply (le_trans (S (Snd x)) (S b) (Snd x)).
Assumption.
Assumption.
Apply Lt_S_n.
Assumption.
div_euclid is defined
Coq <
Coq < Eval Compute in (div_euclid (8) (3)).
     = (Specif.exist nat*nat
         [q:(nat*nat)]
          (8)
           =(Fix plus
               {plus [n:nat] : nat->nat :=
                  [m:nat]
                   Cases n of
                     0 => m
                   | (S p) => (S (plus p m))
                   end} (let (_, y) = q in y)
              (Fix mult
                 {mult [n:nat] : nat->nat :=
                    [m:nat]
                     Cases n of
                       0 => (0)
                     | (S p) =>
                        (Fix plus
                           {plus [n0:nat] : nat->nat :=
                              [m0:nat]
                               Cases n0 of
                                 0 => m0
                               | (S p0) => (S (plus p0 m0))
                               end} m (mult p m))
                     end} (let (x, _) = q in x) (4)))
          /\(le (S (let (_, y) = q in y)) (4)) ((2),(0))
         <((8)=(8)),(le (1) (4))>
          {(<[a:nat](S a)=(8)>
              Cases
                (<[a:nat]a=(7)>
                   Cases
                     (<[a:nat](7)=(S a)>
                        Cases
                          (<[a:nat](6)=(S a)>
```

```
                Cases
                  (<[a:nat](5)=(S a)>
                     Cases
                       (<[a:nat](4)=(S a)>
                          Cases
                            (f_equal2 nat nat nat
                              Fix plus
                                {plus [n:nat]
                                   : nat->nat :=
                                   [m:nat]
                                    Cases n of
                                      0 => m
                                    | (S p) =>
                                       (S (plus p m))
                                    end} (3) (3) (0) (0)
                              (eq_add_S (3) (3)
                                (refl_equal nat (4)))
                              (refl_equal nat (0)))
                          of
                            refl_equal =>
                             (refl_equal nat (4))
                          end)
                     of
                       refl_equal => (refl_equal nat (5))
                     end)
                of
                  refl_equal => (refl_equal nat (6))
                end)
           of
             refl_equal => (refl_equal nat (7))
           end)
      of
        refl_equal => (refl_equal nat (7))
      end)
of
  refl_equal =>
   <[a:nat](8)=(S a)>
     Cases
       (f_equal2 nat nat nat
         Fix plus
           {plus [n:nat] : nat->nat :=
              [m:nat]
               Cases n of
                 0 => m
               | (S p) => (S (plus p m))
```

```
                              end} (3) (3) (4) (4)
                          (eq_add_S (3) (3) (refl_equal nat (4)))
                          (refl_equal nat (4)))
                     of
                       refl_equal => (refl_equal nat (8))
                     end
               end),
            (le_S (1) (3) (le_S (1) (2) (le_S (1) (1) (le_n (1))))))}
     : {q:(nat*nat) |
          ((8)=(plus (Snd q) (mult (Fst q) (4)))/\(lt (Snd q) (4)))}

Coq <
Coq < Eval Compute in (pr1 nat*nat
Coq < [q:nat*nat](8)=(plus (Snd q) (mult (Fst q) (S (3))))/\(lt (Snd q)(S (3)))
Coq <                      (div_euclid (8) (3))).
     = ((2),(0))
     : nat*nat

Coq <
Coq < Eval Compute in (pr2 nat*nat
Coq < [q:nat*nat](8)=(plus (Snd q) (mult (Fst q) (S (3))))/\(lt (Snd q)(S (3)))
Coq <                      (div_euclid (8) (3))).
     = <((8)=(8)),(le (1) (4))>
        {(<[a:nat](S a)=(8)>
            Cases
              (<[a:nat]a=(7)>
                 Cases
                   (<[a:nat](7)=(S a)>
                      Cases
                        (<[a:nat](6)=(S a)>
                           Cases
                             (<[a:nat](5)=(S a)>
                                Cases
                                  (<[a:nat](4)=(S a)>
                                     Cases
                                       (f_equal2 nat nat nat
                                         Fix plus
                                           {plus [n:nat] : nat->nat :=
                                              [m:nat]
                                               Cases n of
                                                 0 => m
                                               | (S p) =>
                                                  (S (plus p m))
                                               end} (3) (3) (0) (0)
                                       (eq_add_S (3) (3)
                                         (refl_equal nat (4)))
```

```
                                        (refl_equal nat (0)))
                                    of
                                      refl_equal =>
                                       (refl_equal nat (4))
                                    end)
                                of
                                  refl_equal => (refl_equal nat (5))
                                end)
                            of
                              refl_equal => (refl_equal nat (6))
                            end)
                        of
                          refl_equal => (refl_equal nat (7))
                        end)
                    of
                      refl_equal => (refl_equal nat (7))
                    end)
                of
                  refl_equal =>
                   <[a:nat](8)=(S a)>
                    Cases
                      (f_equal2 nat nat nat
                        Fix plus
                          {plus [n:nat] : nat->nat :=
                             [m:nat]
                              Cases n of
                                0 => m
                              | (S p) => (S (plus p m))
                              end} (3) (3) (4) (4)
                        (eq_add_S (3) (3) (refl_equal nat (4)))
                        (refl_equal nat (4)))
                    of
                      refl_equal => (refl_equal nat (8))
                    end
                end),
            (le_S (1) (3) (le_S (1) (2) (le_S (1) (1) (le_n (1)))))}
     : ([q:(nat*nat)]
         (8)=(plus (Snd q) (mult (Fst q) (4)))/\(lt (Snd q) (4))
         (pr1 nat*nat
           [q:(nat*nat)]
            (8)=(plus (Snd q) (mult (Fst q) (4)))/\(lt (Snd q) (4))
           (div_euclid (8) (3))))

Coq <
Coq < Definition div_t := [n,m:nat](pr1 nat*nat
```

```
Coq < [q:nat*nat]n=(plus (Snd q) (mult (Fst q) (S m)))/\(lt (Snd q)(S m))
Coq <                     (div_euclid n m)).
div_t is defined

Coq <
Coq < Eval Compute in (div_t (18) (4)).
     = ((3),(3))
     : nat*nat

Coq <
Coq < Eval Compute in (div_t (4) (18)).
     = ((0),(4))
     : nat*nat

Coq <
Coq < Definition div := [n,m:nat;p:~m=0](div_t n (pred m)).
div is defined

Coq <
Coq < Definition Sn_no_0:(n:nat)~(S n)=0.
1 subgoal

  ============================
   (n:nat)~(S n)=(0)

Coq < Intro; Discriminate.
Subtree proved!

Coq < Defined.
Intro; Discriminate.
Sn_no_0 is defined

Coq <
Coq < Check (Sn_no_0 (6)).
(Sn_no_0 (6))
     : ~(7)=(0)

Coq <
Coq < Eval Compute in (div (14) (7) (Sn_no_0 (6))).
     = ((2),(0))
     : nat*nat

Coq <
Coq < Eval Compute in (div (7) (14) (Sn_no_0 (13))).
     = ((0),(7))
     : nat*nat

Coq <
Coq < Theorem Euclides:(a,b:nat)(~b=0) ->
Coq <         {q:(nat*nat) |
```

```
Coq <            (a=(plus (Snd q) (mult (Fst q) b))/\(lt (Snd q) b))}.
1 subgoal

  ============================
   (a,b:nat)
    ~b=(0)
    ->{q:(nat*nat) |
         (a=(plus (Snd q) (mult (Fst q) b))/\(lt (Snd q) b))}

Coq < Proof.

Coq < Destruct b.
2 subgoals

  a : nat
  b : nat
  ============================
   ~(0)=(0)
   ->{q:(nat*nat) |
        (a=(plus (Snd q) (mult (Fst q) (0)))/\(lt (Snd q) (0)))}
subgoal 2 is:
 (n:nat)
  ~(S n)=(0)
  ->{q:(nat*nat) |
       (a=(plus (Snd q) (mult (Fst q) (S n)))/\(lt (Snd q) (S n)))}

Coq < Intros.
2 subgoals

  a : nat
  b : nat
  H : ~(0)=(0)
  ============================
   {q:(nat*nat) |
      (a=(plus (Snd q) (mult (Fst q) (0)))/\(lt (Snd q) (0)))}
subgoal 2 is:
 (n:nat)
  ~(S n)=(0)
  ->{q:(nat*nat) |
       (a=(plus (Snd q) (mult (Fst q) (S n)))/\(lt (Snd q) (S n)))}

Coq < Elim H; Auto.
1 subgoal

  a : nat
  b : nat
  ============================
```

```
   (n:nat)
    ~(S n)=(0)
    ->{q:(nat*nat) |
         (a=(plus (Snd q) (mult (Fst q) (S n)))/\(lt (Snd q) (S n)))}

Coq <
Coq < Intros.
1 subgoal

  a : nat
  b : nat
  n : nat
  H : ~(S n)=(0)
  ============================
   {q:(nat*nat) |
      (a=(plus (Snd q) (mult (Fst q) (S n)))/\(lt (Snd q) (S n)))}

Coq < Apply div_euclid.
Subtree proved!

Coq < Qed.
Destruct b.
Intros.
Elim H; Auto.
Intros.
Apply div_euclid.
Euclides is defined

Coq <
Coq <
Coq <
Coq <
Coq <
```

7

Una táctica más sofisticada: `Inversion`

```
Coq < Inductive par:nat->Prop:=
Coq <        0_par:(par 0)
Coq <  |suma_2_par:(n:nat)(par n)->(par (S (S n))).
Coq < Coq < Coq < Coq < Coq < par is defined
par_ind is defined
```

```
Theorem no_1_par:~(par (1)).
1 subgoal

  ============================
   ~(par (1))

no_1_par < Red.
1 subgoal

  ============================
   (par (1))->False

no_1_par < Intro H.
1 subgoal

  H : (par (1))
  ============================
   False

no_1_par < Elim H;Auto.
1 subgoal

  H : (par (1))
  ============================
   False

no_1_par < Abort.
```

Reintentamos la prueba:

```
Coq < Theorem no_1_par:~(par (1)).
1 subgoal

  ============================
   ~(par (1))

Coq < Proof.

Coq < Red; Intro H.
1 subgoal

  H : (par (1))
  ============================
   False

Coq < Inversion H.
```

```
Subtree proved!

Coq < Qed.
Red; Intro H.
Inversion H.
no_1_par is defined
```
