Capítulo 3: Cálculo de Construcciones Inductivas

4. Inducción y Recursión

Tipos (conjuntos) Inductivos

```
Inductive nat : Set :=
                  0 : nat
                | S : nat → nat
```

```
Inductive bool : Set :=
                  true : bool
                | false : bool
```

```
Inductive natlist : Set :=
                  nil : natlist
                | cons : nat → natlist → natlist
```

Tipos Inductivos Paramétricos

Parámetros: son los argumentos que estan *fijos* y son *globales* en toda una definición.

```
Inductive list (A:Set) : Set :=
                      nil : list A
                    | cons : A → list A → list A
```

Familias Inductivas de Tipos

```
Inductive array(A:Set) : nat → Set :=
          empty : array A 0
        | add : forall n:nat, A → array A n → array A (S n)
```

```
Inductive matrix(A:Set) :  nat→ nat → Set :=
          one_col : forall n:nat, array A n → matrix A 1 n
        | extend_col : forall m n:nat,  matrix A m n → array A n
                                               → matrix A (S m) n
```

Familias Mutuamente Inductivas

```
Inductive
   tree (A:Set) :  Set  :=
               node: A → forest A → tree A
with
   forest (A:Set) : Set  :=
               empty_f : forest A
             | add_tree: tree A → forest A → forest A
```

Predicados Inductivos

```
Inductive Even : nat → Prop :=
            e0 :  Even O
          | eSS : forall n:nat, Even n → Even (S (S n))
```

```
Inductive Le : nat → nat → Prop :=
        le0 : forall n:nat, Le 0 n
      |  leS : forall n m:nat, Le n m → Le (S n) (S m)
```

Definiciones Inductivas - Significado

Cuando escribimos:

$$\textbf{Inductive}\ \mathrm{nat} : \mathrm{Set} :=$$
$$0 : \mathrm{nat}$$
$$\mid\ \ S : \mathrm{nat} \to \mathrm{nat}$$

estamos haciendo muchas cosas…

- estamos definiendo un *conjunto* y una manera de *construir* objetos en él.
- estamos diciendo que ésta es la *única* forma de construir los objetos del conjunto.
- estamos diciendo, además, que las dadas son formas *distintas* de construir objetos.

Definiciones Inductivas - Constructores

- estamos definiendo un conjunto y una manera de construir objetos en él.
  - ➔**A partir de 0 y con reiteradas aplicaciones de la funcion S definimos objetos en nat.**
- estamos diciendo que ésta es la única forma de construir los objetos del conjunto.
  - ➔**Cualquier objeto de nat debe construirse a partir de 0 y con reiteradas aplicaciones de la función S .**
- estamos diciendo, además, que las dadas son formas distintas de construir objetos.
  - ➔**Con 0 y S se construyen objetos *diferentes.* Luego, por ejemplo, $0 \neq (S\ 0)$.**

Consecuencias del significado de una Definición Inductiva (I)

1. Análisis de casos:

Para definir un objeto en un tipo **Q** según un objeto de un tipo inductivo **I** definido con constructores **c_1,...c_n:**

match x with
$c_1 \Rightarrow q_1$
| ...
| $c_n \Rightarrow q_n$
end : Q

Ejemplo: **fun n:nat =>**
match n with
0 $\Rightarrow$ true
| S m $\Rightarrow$ false
end.

es una función de nat $\rightarrow$ bool que decide si un numero es cero o no.

Análisis de casos - Ejemplos

```
Definition pred :=
    fun n:nat => match n with
                    0      ⇒ 0
                  | S m    ⇒ m
                 end.

Definition boolOr :=
    fun b1 b2: bool => match b1 b2 with
                          true  _      ⇒ true
                        | _     true   ⇒ true
                        | false false  ⇒ false
                         end.
```

Análisis de casos dependiente

- El tipo del objeto definido puede también depender del objeto del tipo inductivo sobre el cual se analizan casos:

$$
\begin{array}{l}
\textbf{match } x \textbf{ with} \\
\qquad c_1 \Rightarrow q_1 \\
\qquad \ldots \\
\qquad c_n \Rightarrow q_n \\
\qquad \textbf{end: } Q\ x
\end{array}
$$

- Tiene más sentido en el contexto de definiciones recursivas y pruebas de propiedades (más adelante)

Análisis de casos en Coq - Tácticas

- Aplicación de constructores:
  - apply c_i
  - constructor i (= intros ; apply c_i)/ constructor
- Discriminación de constructores:
  - discriminate H (prueba cualquier cosa si H: t_1=t_2, con t_1 y t_2 construídos con distintos constructores)
- Inyectividad de constructores:
  - injection H (saca constructores iguales de una igualdad)
- En general...
  - simplify_eq (aplica discriminate o injection)

Consecuencias del significado de una Definición Inductiva (II)

2. Recursión:

Para definir recursivamente un objeto en un tipo **Q** haciendo recursión en un objeto de un tipo inductivo **I.**

$$\text{Fixpoint } f\ (x_1 : I_1) \dots (x_n : I_n) : Q := e$$

La expresión e en general será un `match` en algún x_i, y podrá contener a f bajo ciertas condiciones sintácticas* que aseguran la ***terminación***.

Estas condiciones se chequean sobre el ***último argumento*** de la lista $x_1 : I_1, \dots x_n : I_n$.

*Para que un llamado recursivo sea correcto debe realizarse sobre un elemento ***estructuralmente más pequeño***

Recursión - Ejemplos

Recursión en el primer argumento

```
Fixpoint add (n m:nat) {struct n}:  nat  :=
 match n with
          0   ⇒  m
    |   S k ⇒  S (add k m)
end.
```

Recursión en el segundo argumento

```
Fixpoint add (n m:nat) {struct m}:  nat  :=
 match m with
          0   ⇒  n
    |   S k ⇒  S (add n k)
end.
```

Recursión - Más ejemplos

```
Fixpoint even (n:nat) : bool :=
   match n with
        0  ⇒ true
   |  S k  ⇒  match k with
                   0    ⇒ false
               |   S m ⇒ even m
               end
   end.

Fixpoint mod2 (n:nat) : nat :=
   match n with
      0    ⇒  0
   |  S 0  ⇒ S 0
   |  S (S m) ⇒ mod2 m
   end.
```

Recursión Mutua - Ejemplos

```
Fixpoint even (n:nat) : bool :=
        match n with   0 ⇒  true |  S k ⇒ odd k          end
with  odd (n:nat) : bool :=
        match n with   0 ⇒  false |  S k ⇒ even k        end.


Fixpoint tree_size (t:tree) : nat :=
   match t with      (node a f ) ⇒ S (forest_size f )    end
with forest_size (f:forest) : nat :=
   match f with
            empty_f  ⇒  0
        |   add_tree t x  ⇒ plus (tree_size t )(forest_size x )
   end.
```

Consecuencias del significado de una Definición Inductiva (III)

3. Análisis de casos

Para probar una propiedad P (: Prop) por casos en un objeto x de un tipo inductivo I

Tácticas:

- case x: genera los casos según la definición de I.
- destruct x: aplica intros y después case

Pruebas por casos en Coq

Ejemplos

$$\frac{\Gamma \quad n: nat}{P\ n} \quad \text{case n} \quad \frac{\Gamma}{P\ 0} \quad \frac{\Gamma}{\text{forall } x{:}nat, P\ (S\ x)}$$

$$\frac{\Gamma}{\text{forall } n{:}\ nat, P\ n} \quad \text{destruct n} \quad \frac{\Gamma \quad n{:}nat}{P\ 0} \quad \frac{\Gamma \quad n{:}nat}{\text{forall } x{:}\ nat,\ P\ (S\ x)}$$

Consecuencias del significado de una Definición Inductiva (IV)

4. Inducción

Para probar una propiedad P utilizando el ***principio de inducción primitiva*** asociado a la definición inductiva de un tipo I.

Tácticas:

- elim x: genera los casos según la definición de x, *con sus hipótesis inductivas*
- induction x: aplica intros antes y después elim

Pruebas por Inducción en Coq
Ejemplos

$$\frac{\begin{array}{l}\Gamma\\ n:\ nat\end{array}}{P\ n} \quad \text{elim n} \quad \frac{\begin{array}{l}\Gamma\\ n:nat\end{array}}{P\ 0} \qquad \frac{\begin{array}{l}\Gamma\\ n:nat\end{array}}{\text{forall } x:nat, P\ x \rightarrow P(S\ x)}$$

$$\frac{\begin{array}{l}\Gamma\\ n:\ nat\\ e:\ Even\ n\end{array}}{P\ n} \quad \text{elim e} \quad \frac{\begin{array}{l}\Gamma\\ n:\ nat\\ e:\ Even\ n\end{array}}{P\ 0} \qquad \frac{\begin{array}{l}\Gamma\\ n:nat\\ e:\ Even\ n\end{array}}{\begin{array}{l}\text{forall } x:nat, Even\ x \rightarrow\\ P\ x \rightarrow P\ (S\ (S\ x))\end{array}}$$

Destructores

- Cuando se define un tipo inductivo **I**, Coq genera tres constantes correspondientes a los principios de recursión e inducción:
  - **I_ind es el principio de inducción para Prop**
    - *implementa el principio de inducción estructural para objetos de I*
  - **I_rec es el principio de inducción para Set**
    - *permite hacer definiciones recursivas sobre objetos de I*
  - **I_rect es el principio de inducción para Type**
    - *permite definir familias recursivas de tipos*

ver reference manual secc 1.3.3

Destructores - Ejemplos

Inductive nat : Set := 0 : nat
| S : nat → nat

---

nat_ind: forall P:nat→Prop,
P 0 → (forall x:nat,P x → P (S x)) → forall n:nat,P n

---

nat_rec: forall A:nat→Set,
A 0 → (forall x:nat,A x → A (S x)) → forall n:nat,A n

---

nat_rect: forall T:nat→Type,
T 0 → (forall x:nat,T x → T (S x)) → forall n:nat,T n

Destructores - Ejemplos

Inductive list (A:Set) : Set :=
nil : list A
| cons : A $\rightarrow$ list A $\rightarrow$ list A

---

list_ind: forall A:set, forall P: list A $\rightarrow$Prop,
P (nil A) $\rightarrow$
(forall (a:A)(x: list A) P x $\rightarrow$ P (cons A a x)) $\rightarrow$
forall l: list A, P l

Destructores - Ejemplos

Inductive array (A:Set) : nat $\rightarrow$ Set :=

empty : array A 0

| add : forall n:nat, A $\rightarrow$ array A n $\rightarrow$ array A (S n)

---

array_ind: forall (A : set) (P : forall n:nat, array A n $\rightarrow$Prop)

P 0 (empty A) $\rightarrow$

(forall (n:nat)(a:A)(x: array A n), P n x $\rightarrow$ P (S n) (add A n a x))

$\rightarrow$ forall (n:nat)(v: array A n), P n v

Destructores - Ejemplos

Inductive Even : nat → Prop :=

e0 : Even O

| eSS : forall n:nat, Even n → Even (S (S n))

Even_ind: forall P:nat→Prop,

P 0 →

(forall x:nat, Even x → P x → P (S (S x))) →

forall n:nat, Even n → P n

Tácticas y Destructores

elim = apply el destructor correspondiente

$$\frac{\Gamma \quad n:\text{nat}}{A\ n} \quad \textbf{elim n} \quad \frac{\Gamma \quad n:\text{nat}}{A\ 0} \quad \frac{\Gamma \quad n:\text{nat}}{\text{forall } x:\text{nat}, A\ x \to A(S\ x)}$$

(= apply nat_rec)

$$\frac{\Gamma \quad n:\text{nat} \quad e:\text{Even } n}{P\ n} \quad \textbf{elim H} \quad \frac{\Gamma \quad n:\text{nat} \quad e:\text{Even } n}{P\ 0} \quad \frac{\Gamma \quad n:\text{nat} \quad e:\text{Even } n}{\text{forall } x:\text{nat}, \text{Even } x \to P\ x \to P(S(S\ x))}$$

(= apply Even_ind)

Conectivos: Definición Inductiva

- Los conectivos $\wedge$, $\vee$, $\perp$ y $\exists$ se definen como constructores de proposiciones inductivos:

Inductive and (A B:Prop) : Prop :=
conj: A $\rightarrow$ B $\rightarrow$ and A B

Inductive or (A B:Prop) : Prop :=
or_introl: A $\rightarrow$ or A B
| or_intror: B $\rightarrow$ or A B

Conectivos - Destructores

Inductive and (A B:Prop) : Prop := conj: A $\to$ B $\to$ and A B

➔ and_ind: forall A B P:Prop), (A $\to$ B $\to$ P) $\to$ (A $\wedge$ B $\to$ P)

Inductive or (A B:Prop) : Prop :=

or_introl: A $\to$ or A B

| or_intror: B $\to$ or A B

➔ or_ind: forall A B P:Prop, (A$\to$ P)$\to$ (B$\to$ P)$\to$ (A$\vee$ B$\to$ P)

Más Conectivos...

Inductive True: Prop := I : True

➔True_ind: forall P: Prop, P $\rightarrow$ True $\rightarrow$ P

Inductive False: Prop :=

➔False_ind: forall P: Prop, False $\rightarrow$ P

Inductive eq (A:Set)(x:A): A$\rightarrow$Prop :=
refl_equal: eq A x x

➔eq_ind: forall (A:Set) (x:A) (P: A$\rightarrow$Prop), (P x) $\rightarrow$
(forall y:A, eq x y) $\rightarrow$ (P y))

Cuantificador Existencial

Inductive ex (A:Set)(P:A$\rightarrow$Prop) : Prop :=

ex_intro: forall x:A, P x $\rightarrow$ ex A P

➔ex_ind: forall (A:Set) (P: A$\rightarrow$Prop) (Q:Prop),

(forall x:A, P x $\rightarrow$ Q) $\rightarrow$ (ex A P $\rightarrow$ Q)

Cálculo de Construcciones Inductivas

Sintaxis

T := Set | Type | Prop

| x — variables

| c — constantes definidas

| (T T) — aplicación

| [x : T]T — abstracción

| (x : T) T — producto

| T → T — tipo de las funciones

| Inductive x [x:T,..x:T] : T := c:T | …| c:T — def. ind.

| <T> match T with T⇒T |...| T⇒ T end — def. casos

| Fixpoint x [x:T,..x:T] : T := T — def. recursiva

Notaciones en el lenguaje matemático
Elementos canónicos y no canónicos

- **Elementos canónicos de un tipo:**
  - aquellos cuyo significado es primitivo (valores).
  - Ej: 0, (S 0), (S (S 0)), son elementos canónicos de nat
- **Elementos no canónicos de un tipo:**
  - son notaciones o abreviaturas.
  - Tienen significado únicamente si denotan un elemento canónico.
  - Para conocer su significado deben desabreviarse.
  - Ej: (S 0)+0, (S 0)+(S (S 0)), (pred 0), son elementos NO canónicos de nat

Elementos canónicos y no canónicos

- **Constructores del tipo inductivo I:** son métodos para construir los elementos **canónicos** del tipo.

- **Eliminadores del tipo inductivo I:** permiten construir elementos **no canónicos** de un tipo Q

  (Q puede ser igual a I, o incluso Q puede ser Prop).

- Regla de cálculo asociada a los eliminadores (o destructores) de tipos inductivos : $\longrightarrow\iota$
  - se aplica cuando el ***destructor*** está aplicado a un término en forma de ***constructor***.

Reducción Iota - Ejemplos

pred 0

$\rightarrow_\delta$ **fun n:nat => match n with 0 $\Rightarrow$ 0 | S m $\Rightarrow$ m end 0**

$\rightarrow\beta$ **match 0 with 0 $\Rightarrow$ 0 | (S m) $\Rightarrow$ m end**

$\rightarrow\iota$ 0

pred (S (S 0))

$\rightarrow_\delta$ **fun n:nat => match n with 0 $\Rightarrow$ 0 | S m $\Rightarrow$ m end (S (S 0))**

$\rightarrow\beta$ **match (S (S 0)) with 0 $\Rightarrow$ 0 | (S m) $\Rightarrow$ m end**

$\rightarrow\iota$ (S 0)