

Inducción sobre booleanos

El principio de reemplazo no alcanza para probar todas las equivalencias que nos interesan.

Ejemplo

$\{NT\}$ `not True = False`

$\{NF\}$ `not False = True`

¿Podemos probar $\forall x :: \text{Bool}. \text{not } (\text{not } x) = x$?

El problema es que la expresión

$$\text{not } (\text{not } x)$$

está “trabada”: no se puede aplicar ninguna ecuación.

Inducción sobre booleanos

Principio de inducción sobre booleanos

Si $\mathcal{P}(\text{True})$ y $\mathcal{P}(\text{False})$ entonces $\forall x :: \text{Bool}. \mathcal{P}(x)$.

Ejemplo

$\{\text{NT}\}$ not True = False

$\{\text{NF}\}$ not False = True

Para probar $\forall x :: \text{Bool}. \text{not} (\text{not } x) = x$
basta probar:

1. not (not True) = True.

$$\text{not} (\text{not True}) = \text{not False} = \text{True}$$

\uparrow \uparrow
NT NF

2. not (not False) = False.

$$\text{not} (\text{not False}) = \text{not True} = \text{False}$$

\uparrow \uparrow
NF NT

Inducción sobre pares

Cada tipo de datos tiene su propio principio de inducción.

Ejemplo

{FST} $\text{fst } (x, _) = x$

{SND} $\text{snd } (_, y) = y$

{SWAP} $\text{swap } (x, y) = (y, x)$

¿Podemos probar $\forall p :: (a, b). \text{fst } p = \text{snd } (\text{swap } p)$?

Las expresiones $(\text{fst } p)$ y $(\text{snd } (\text{swap } p))$ están “trabadas”.

Inducción sobre pares

Principio de inducción sobre pares

Si $\forall x :: a. \forall y :: b. \mathcal{P}((x, y))$

entonces $\forall p :: (a, b). \mathcal{P}(p)$.

Ejemplo

{FST} $\text{fst } (x, _) = x$

{SND} $\text{snd } (_, y) = y$

{SWAP} $\text{swap } (x, y) = (y, x)$

Para probar $\forall p :: (a, b). \text{fst } p = \text{snd } (\text{swap } p)$

basta probar:

► $\forall x :: a. \forall y :: b. \text{fst } (x, y) = \text{snd } (\text{swap } (x, y))$

$$\text{fst } (x, y) \underset{\substack{\uparrow \\ \text{FST}}}{=} x \underset{\substack{\uparrow \\ \text{SND}}}{=} \text{snd } (y, x) \underset{\substack{\uparrow \\ \text{SWAP}}}{=} \text{snd } (\text{swap } (x, y))$$

Inducción sobre naturales

```
data Nat = Zero | Suc Nat
```

Principio de inducción sobre naturales

Si $\mathcal{P}(\text{Zero})$ y $\forall n :: \text{Nat. } (\underbrace{\mathcal{P}(n)}_{\text{hipótesis inductiva}} \Rightarrow \underbrace{\mathcal{P}(\text{Suc } n)}_{\text{tesis inductiva}})$,
entonces $\forall n :: \text{Nat. } \mathcal{P}(n)$.

Inducción sobre naturales

Ejemplo

$\{S0\}$ suma Zero m = m

$\{S1\}$ suma (Suc n) m = Suc (suma n) m

Para probar $\forall n :: \text{Nat. suma n Zero} = n$

basta probar:

1. suma Zero Zero = Zero.

Inmediato por S0.

2. $\underbrace{\text{suma n Zero} = n}_{\text{H.I.}} \Rightarrow \underbrace{\text{suma (Suc n) Zero} = \text{Suc n}}_{\text{T.I.}}$

$$\text{suma (Suc n) Zero} = \underset{\substack{\uparrow \\ \text{S1}}}{\text{Suc}} \text{ (suma n Zero)} = \underset{\substack{\uparrow \\ \text{H.I.}}}{\text{Suc}} \text{ n}$$

Inducción estructural

En el **caso general**, tenemos un tipo de datos inductivo:

$$\begin{array}{lcl} \text{data } T & = & \text{CBase}_1 \langle \text{parámetros} \rangle \\ & & \dots \\ & | & \text{CBase}_n \langle \text{parámetros} \rangle \\ & | & \text{CRecursoivo}_1 \langle \text{parámetros} \rangle \\ & & \dots \\ & | & \text{CRecursoivo}_m \langle \text{parámetros} \rangle \end{array}$$

Principio de inducción estructural

Sea \mathcal{P} una propiedad acerca de las expresiones tipo T tal que:

- ▶ \mathcal{P} vale sobre todos los constructores base de T ,
- ▶ \mathcal{P} vale sobre todos los constructores recursivos de T ,
asumiendo como hipótesis inductiva que vale para los
parámetros de tipo T ,

entonces $\forall x :: T. \mathcal{P}(x)$.

Inducción estructural

Ejemplo: principio de inducción sobre listas

data [a] = [] | a : [a]

Sea \mathcal{P} una propiedad sobre expresiones de tipo [a] tal que:

- ▶ $\mathcal{P}([])$
- ▶ $\forall x :: a. \forall xs :: [a]. \underbrace{(\mathcal{P}(xs))}_{\text{H.I.}} \Rightarrow \underbrace{\mathcal{P}(x : xs)}_{\text{T.I.}}$

Entonces $\forall xs :: [a]. \mathcal{P}(xs)$.

Ejemplo: principio de inducción sobre árboles binarios

data AB a = Nil | Bin (AB a) a (AB a)

Sea \mathcal{P} una propiedad sobre expresiones de tipo AB a tal que:

- ▶ $\mathcal{P}(\text{Nil})$
- ▶ $\forall i :: AB a. \forall r :: a. \forall d :: AB a. \underbrace{((\mathcal{P}(i) \wedge \mathcal{P}(d)))}_{\text{H.I.}} \Rightarrow \underbrace{\mathcal{P}(\text{Bin } i \ r \ d))}_{\text{T.I.}}$

Entonces $\forall x :: AB a. \mathcal{P}(x)$.

Inducción estructural

Ejemplo: principio de inducción sobre polinomios

```
data Poli a = X
            | Cte a
            | Suma (Poli a) (Poli a)
            | Prod (Poli a) (Poli a)
```

Sea \mathcal{P} una propiedad sobre expresiones de tipo `Poli a` tal que:

- ▶ $\mathcal{P}(X)$
- ▶ $\forall k :: a. \mathcal{P}(\text{Cte } k)$
- ▶ $\forall p :: \text{Poli } a. \forall q :: \text{Poli } a.$
$$\underbrace{((\mathcal{P}(p) \wedge \mathcal{P}(q))}_{\text{H.I.}} \Rightarrow \underbrace{\mathcal{P}(\text{Suma } p \text{ } q))}_{\text{T.I.}}$$
- ▶ $\forall p :: \text{Poli } a. \forall q :: \text{Poli } a.$
$$\underbrace{((\mathcal{P}(p) \wedge \mathcal{P}(q))}_{\text{H.I.}} \Rightarrow \underbrace{\mathcal{P}(\text{Prod } p \text{ } q))}_{\text{T.I.}}$$

Entonces $\forall x :: \text{Poli } a. \mathcal{P}(x)$.

Ejemplo: inducción sobre listas

$$\{M0\} \text{ map } f [] = []$$

$$\{M1\} \text{ map } f (x : xs) = f x : \text{map } f xs$$

$$\{A0\} [] ++ ys = ys$$

$$\{A1\} (x : xs) ++ ys = x : (xs ++ ys)$$

Propiedad. Si $f :: a \rightarrow b$, $xs :: [a]$, $ys :: [a]$, entonces:

$$\text{map } f (xs ++ ys) = \text{map } f xs ++ \text{map } f ys$$

Por inducción en la estructura de xs , basta ver:

1. Caso base, $\mathcal{P}([])$.

2. Caso inductivo, $\forall x :: a. \forall xs :: [a]. (\mathcal{P}(xs) \Rightarrow \mathcal{P}(x : xs))$.

con $\mathcal{P}(xs) :\equiv (\text{map } f (xs ++ ys) = \text{map } f xs ++ \text{map } f ys)$.

Ejemplo: inducción sobre listas

Caso base:

```
map f ([] ++ ys)
= map f ys                por A0
= [] ++ map f ys          por A0
= map f [] ++ map f ys    por M0
```

Caso inductivo:

```
map f ((x : xs) ++ ys)
= map f (x : (xs ++ ys))    por A1
= f x : map f (xs ++ ys)    por M1
= f x : (map f xs ++ map f ys) por H.I.
= (f x : map f xs) ++ map f ys por A1
= map f (x : xs) ++ map f ys por M1
```

Ejemplo: relación entre foldr y foldl

Propiedad. Si $f :: a \rightarrow b \rightarrow b$, $z :: b$, $xs :: [a]$, entonces:

$$\underbrace{\text{foldr } f \ z \ xs = \text{foldl } (\text{flip } f) \ z \ (\text{reverse } xs)}_{\mathcal{P}(xs)}$$

Por inducción en la estructura de xs . El caso base $\mathcal{P}([])$ es fácil.

Caso inductivo, $\forall x :: a. \forall xs :: [a]. (\mathcal{P}(xs) \Rightarrow \mathcal{P}(x : xs))$:

$$\begin{aligned} & \text{foldr } f \ z \ (x : xs) \\ = & f \ x \ (\text{foldr } f \ z \ xs) && (\text{Def. foldr}) \\ = & f \ x \ (\text{foldl } (\text{flip } f) \ z \ (\text{reverse } xs)) && (\text{H.I.}) \\ = & \text{flip } f \ (\text{foldl } (\text{flip } f) \ z \ (\text{reverse } xs)) \ x && (\text{Def. flip}) \\ = & \text{foldl } (\text{flip } f) \ z \ (\text{reverse } xs ++ [x]) && \textcolor{red}{(???) } \\ = & \text{foldl } (\text{flip } f) \ z \ (\text{reverse } (x : xs)) && (\text{Def. reverse}) \end{aligned}$$

Para justificar el paso faltante $\textcolor{red}{(???)}$, se puede demostrar:

Lema. Si $g :: b \rightarrow a \rightarrow b$, $z :: b$, $x :: a$, $xs :: [a]$, entonces:

$$\text{foldl } g \ z \ (xs ++ [x]) = g \ (\text{foldl } g \ z \ xs) \ x$$

Introducción

Inducción estructural

Extensionalidad

Isomorfismos de tipos

Extensionalidad

Usando el principio de inducción estructural, se puede probar:

Extensionalidad para pares

Si $p :: (a, b)$, entonces $\exists x :: a. \exists y :: b. p = (x, y)$.

```
data Either a b = Left a | Right b
```

Extensionalidad para sumas

Si $e :: \text{Either } a \ b$, entonces:

- ▶ o bien $\exists x :: a. e = \text{Left } x$
- ▶ o bien $\exists y :: b. e = \text{Right } y$

Puntos de vista intensional vs. extensional

¿Vale la siguiente equivalencia de expresiones?

```
quickSort = insertionSort
```

Depende del punto de vista.

- ▶ **Punto de vista intensional.** (va con “s”)
Dos valores son iguales si están definidos de la misma manera.
- ▶ **Punto de vista extensional.**
Dos valores son iguales si son indistinguibles al observarlos.

Ejemplo

`quickSort` e `insertionSort`

- ▶ **no** son **intensionalmente** iguales;
- ▶ **sí** son **extensionalmente** iguales: computan la misma función.

Principio de extensionalidad funcional

Sean $f, g :: a \rightarrow b$.

Propiedad inmediata

Si $f = g$ entonces $(\forall x :: a. f\ x = g\ x)$.

Principio de extensionalidad funcional

Si $(\forall x :: a. f\ x = g\ x)$ entonces $f = g$.

Principio de extensionalidad funcional

Ejemplo: extensionalidad funcional

$\{I\}$ $\text{id } x = x$ $\{C\}$ $(g \ . \ f) \ x = g \ (f \ x)$

$\{S\}$ $\text{swap } (x, y) = (y, x)$

Veamos que $\text{swap} \ . \ \text{swap} = \text{id} :: (a, b) \rightarrow (a, b)$.

Por extensionalidad funcional, basta ver:

$$\forall p :: (a, b). (\text{swap} \ . \ \text{swap}) \ p = \text{id} \ p$$

Por inducción sobre pares, basta ver:

$$\forall x :: a. \ \forall y :: b. (\text{swap} \ . \ \text{swap}) \ (x, y) = \text{id} \ (x, y)$$

En efecto:	$(\text{swap} \ . \ \text{swap}) \ (x, y)$	
	$= \text{swap} \ (\text{swap} \ (x, y))$	(por C)
	$= \text{swap} \ (y, x)$	(por S)
	$= (x, y)$	(por S)
	$= \text{id} \ (x, y)$	(por I)

Resumen: razonamiento ecuacional

Razonamos ecuacionalmente usando tres principios:

1. **Principio de reemplazo**

Si el programa declara que $e1 = e2$, cualquier instancia de $e1$ es igual a la correspondiente instancia de $e2$, y viceversa.

2. **Principio de inducción estructural**

Para probar \mathcal{P} sobre todas las instancias de un tipo T , basta probar \mathcal{P} para cada uno de los constructores (asumiendo la H.I. para los constructores recursivos).

3. **Principio de extensionalidad funcional**

Para probar que dos funciones son iguales, basta probar que son iguales punto a punto.

Corrección del razonamiento ecuacional

Supongamos que logramos demostrar que $e1 = e2$.

¿Qué nos asegura eso sobre $e1$ y $e2$?

Cuidado: no necesariamente dan el mismo resultado

Por ejemplo, se puede demostrar:

```
quickSort = insertionSort
```

pero `quickSort` e `insertionSort` no dan el mismo resultado.

Corrección con respecto a observaciones

Si demostramos $e1 = e2 :: A$, entonces:

$$\text{obs } e1 \rightsquigarrow \text{True} \quad \text{si y sólo si} \quad \text{obs } e2 \rightsquigarrow \text{True}$$

para toda posible “observación” $\text{obs} :: A \rightarrow \text{Bool}$.

Demostración de desigualdades

¿Cómo demostramos que **no** vale una igualdad $e1 = e2 :: A$?

Por la contrarrecíproca de la anterior, basta con encontrar una observación $obs :: A \rightarrow Bool$ que las distinga.

Ejemplo

Demostrar que **no** vale la igualdad:

$id = swap :: (Int, Int) \rightarrow (Int, Int)$

$obs :: ((Int, Int) \rightarrow (Int, Int)) \rightarrow Bool$

$obs\ f = fst\ (f\ (1, 2)) == 1$

$obs\ id \rightsquigarrow True$

$obs\ swap \rightsquigarrow False$

Introducción

Inducción estructural

Extensionalidad

Isomorfismos de tipos

Misma información, distinta forma

¿Qué relación hay entre los siguientes valores?

```
("hola", (1, True)) :: (String, (Int, Bool))
```

```
((True, "hola"), 1) :: ((Bool, String), Int)
```

Representan la misma información, pero escrita de distinta manera.

Podemos transformar los valores de un tipo en valores del otro:

```
f :: (String, (Int, Bool)) -> ((Bool, String), Int)
```

```
f (s, (i, b)) = ((b, s), i)
```

```
g :: ((Bool, String), Int) -> (String, (Int, Bool))
```

```
g ((b, s), i) = (s, (i, b))
```

Se puede demostrar que:

$$g \cdot f = \text{id} \qquad f \cdot g = \text{id}$$

Isomorfismos de tipos

Definición

Decimos que dos tipos de datos A y B son **isomorfos** si:

1. Hay una función $f :: A \rightarrow B$ total.
2. Hay una función $g :: B \rightarrow A$ total.
3. Se puede demostrar que $g \circ f = \text{id} :: A \rightarrow A$.
4. Se puede demostrar que $f \circ g = \text{id} :: B \rightarrow B$.

Escribimos $A \simeq B$ para indicar que A y B son isomorfos.

Ejemplo de isomorfismo: currificación

Ejemplo

Veamos que $((a, b) \rightarrow c) \simeq (a \rightarrow b \rightarrow c)$.

```
curry :: ((a, b) -> c) -> a -> b -> c
curry f x y = f (x, y)
```

```
uncurry :: (a -> b -> c) -> (a, b) -> c
uncurry f (x, y) = f x y
```


Ejemplo de isomorfismo: currificación

Veamos que

$\text{uncurry} \cdot \text{curry} = \text{id} \quad :: ((a, b) \rightarrow c) \rightarrow (a, b) \rightarrow c$

Por extensionalidad funcional, basta ver que si $f :: (a, b) \rightarrow c$:

$(\text{uncurry} \cdot \text{curry}) f = \text{id} f \quad :: (a, b) \rightarrow c$

Por extensionalidad funcional, basta ver que si $p :: (a, b)$:

$(\text{uncurry} \cdot \text{curry}) f p = \text{id} f p \quad :: c$

Por inducción sobre pares, basta ver que si $x :: a, y :: b$:

$(\text{uncurry} \cdot \text{curry}) f (x, y) = \text{id} f (x, y) \quad :: c$

En efecto:

$$\begin{aligned} & (\text{uncurry} \cdot \text{curry}) f (x, y) \\ = & \text{uncurry} (\text{curry} f) (x, y) && (\text{Def. } (..)) \\ = & \text{curry} f x y && (\text{Def. uncurry}) \\ = & f (x, y) && (\text{Def. curry}) \\ = & \text{id} f (x, y) && (\text{Def. id}) \end{aligned}$$

(Y vale también $\text{curry} \cdot \text{uncurry} = \text{id}$).

Más isomorfismos de tipos

$$(a, b) \quad \simeq \quad (b, a)$$

$$(a, (b, c)) \quad \simeq \quad ((a, b), c)$$

$$a \rightarrow b \rightarrow c \quad \simeq \quad b \rightarrow a \rightarrow c$$

$$a \rightarrow (b, c) \quad \simeq \quad (a \rightarrow b, a \rightarrow c)$$

$$\text{Either } a \ b \rightarrow c \quad \simeq \quad (a \rightarrow c, b \rightarrow c)$$