Ejercicio 1 ★

Sean las siguientes definiciones de funciones:

```
- intercambiar (x,y) = (y,x) I
- espejar (Left x)  = Right x
  espejar (Right x) = Left x
- asociarI (x,(y,z)) = ((x,y),z)
- asociarD ((x,y),z)) = (x,(y,z))
- flip f x y = f y x
- curry f x y = f (x,y)
- uncurry f (x,y) = f x y
```

Demostrar las siguientes igualdades usando los lemas de generación cuando sea necesario:

- I. ∀ p::(a,b) . intercambiar (intercambiar p) = p
- II. ∀ p::(a,(b,c)) . asociarD (asociarI p) = p
- III. ∀ p::Either a b . espejar (espejar p) = p
- IV. ∀ f::a->b->c . ∀ x::a . ∀ y::b . flip (flip f) x y = f x y
- V. ∀ f::a->b->c . ∀ x::a . ∀ y::b . curry (uncurry f) x y = f x y

i) Por el lema de gen de pares $(\forall p:(x,y))(\exists x:a, y:b) \cdot (p = (x,y))$

INTERCAMBIAR (INTERCAMBIAR (x,y)) = (x,y)

POR PPIO DE REEMPLAZO

{I} INTERCAMBIAR $(\underline{y,x}) \overset{\{I\}}{=} (x,y)$

no imp el orden de params (nombre)

Luego, la igualdad vale ->¿TENGO Q ACLARAR ALGO MÁS?

ii) Por el lema de gen de pares.

$(\forall p::(x,(y,z)))(\exists x::a, y::b, z::c)$ (ASOCIARD (ASOCIARI (x,(y,z)) = (x,(y,z)))

Por el PPIO DE REEMPLAZO

$\overset{AI}{=}$ ASOCIARD((x,y),z) $\overset{AD}{=}$ (x,(y,z)) =

Luego, la igualdad vale ->¿TENGO Q ACLARAR ALGO MÁS?

iii) Por el lema de generación de sumas

$(\forall p::\text{EITHER } x\ y)(\exists x::a, y::b)$ tq P = LEFT X ó P = RIGHT Y

Tenemos dos casos

- CASO P = LEFT X

  ESPEJAR (ESPEJAR (LEFT X)) = LEFT X

  $\overset{\{EL\}}{=}$ ESPEJAR (RIGHT X) $\overset{\{ER\}}{=}$ LEFT X

- CASO P = RIGHT X

  ESPEJAR(ESPEJAR (RIGHT X)) = RIGHT X =

  $\overset{\{ER\}}{=}$ ESPEJAR (LEFT X) $\overset{\{EL\}}{=}$ RIGHT X

Luego, la igualdad vale

V) $\forall f::a\to b\to c.\ \forall x::a.\ \forall y::b(\text{CURRY}(\text{UNCURRY } f)\ x\ y = f\ x\ y)$

POR PPIO DE REEMPLAZO

$(\text{CURRY}(\text{UNCURRY } f))\ x\ y \overset{\{C\}}{=} \text{UNCURRY } f\ (x,y) \overset{\{U\}}{=} f\ x\ y$

Luego, la igualdad vale

Ejercicio 2 ★

Demostrar las siguientes igualdades utilizando el principio de extensionalidad funcional:

- I. `flip . flip = id`
- II. ∀ `f::(a,b)->c . uncurry (curry f) = f`
- III. `flip const = id`
- IV. ∀ `f::a->b` . ∀ `g::b->c` . ∀ `h::c->d . ((h . g) . f) = (h . (g . f))` con la definición usual de la composición: `(.) f g x = f (g x)`.

¿SIRVE PARA HACER APARECER ARGS? Si

DA FORMA

LEMA GENERACIÓN: ∃

EXT FUNCIONALIDAD: ∀

↳ SPAWNEA COSAS.

i)

$\text{FLIP}::(a\to b\to c)\to(b\to a\to c)$

Ahora por PPIO. de EXT funcional $\forall f: a\to b\to c$ vale que $(\text{FLIP}.\text{FLIP})\ f = \text{ID}\ f$

$(\text{FLIP}.\text{FLIP})\ f = \text{ID}\ f$

$\overset{\{.\}}{=} \text{FLIP}(\text{FLIP } f) = \text{ID}\ f$

Ahora por PPIO. de EXT funcional $\forall x::a$ vale que $\text{FLIP}(\text{FLIP } f)\ x = \text{ID}\ f\ x$

Ahora por PPIO. de EXT funcional $\forall y::b$ vale que $\text{FLIP}(\text{FLIP } f)\ x\ y = \text{ID}\ f\ x\ y$

$\text{FLIP}(\text{FLIP } f)\ x\ y = (\text{ID}\ f)\ x\ y$

$\text{FLIP}\ f\ y\ x \overset{\{\text{FLIP}\}}{=} (\text{ID}\ f)\ x\ y$

$\text{FLIP}\ f\ y\ x \overset{\{\text{ID}\}}{=} f\ x\ y$

$f\ x\ y \overset{\{\text{FLIP}\}}{=} f\ x\ y$

Como queríamos demostrar.

b) $\forall f :: (a,b) \to c \cdot$ UNCURRY (CURRY f) = f

POR PPIO DE EXT FUNCIONAL: $(\forall p : (a,b))(\forall f :: (a,b) \to c)$ (UNCURRY (CURRY f) p = f p)

POR LEMA DE GENERACIÓN DE PARES, $(\exists x : a, y : b)$ $(p = (x,y))$

UNCURRY (CURRY f) (x,y) = $f(x,y)$

$\stackrel{\{UC\}}{=}$ CURRY f x y = $f(x,y)$

$\stackrel{\{C\}}{=}$ $f(x,y) = f(x,y)$

Como queríamos demostrar

C) FLIP CONST = CONST ID

FLIP CONST : $b \to a \to a$

POR EL PPIO DE EXT FUNCIONAL $(\forall x : a)$ (FLIP CONST x = (CONST ID x))

FLIP :: $(a \to b \to c) \to b \to a \to c$

POR EL PPIO DE EXT FUNCIONAL $(\forall x : a)(\forall y : b)$ (FLIP CONST x y = (CONST ID x) y)

CONST :: $a \to b \to a$

FLIP CONST x y = CONST ID x y

ID :: $a \to a$

FLIP CONST x y $\stackrel{\{C\}}{=}$ ID y

CONST y x $\stackrel{\{F\}}{=}$ ID y

y $\stackrel{\{C; ID\}}{=}$ y

d) POR EXT FUNCIONAL $\forall f :: a \to b, \forall g :: b \to c, \forall h :: c \to d$. $(h \cdot g) \cdot f = h \cdot (g \cdot f)$

$((h \cdot g) \cdot f)\ x = (h \cdot (g \cdot f))\ x$

$\stackrel{\{.\}}{=}$ $(h \cdot g)\ (f\ x)$

$\stackrel{\{.\}}{=}$ $h\ (g\ (f\ x))$ $\quad\stackrel{\{.\}}{=}$ $(h \cdot g)\ (f\ x)$

$= h\ (g\ (f\ x))$ $\quad\stackrel{\{.\}}{=}$ $h\ (g\ (f\ x))$

Demostrar las siguientes propiedades:

- I. ∀ xs::[a] . length (duplicar xs) = 2 * length xs
- II. ∀ xs::[a] . ∀ ys::[a] . length (xs ++ ys) = length xs + length ys
- III. ∀ xs::[a] . ∀ x::a . append [x] xs = x:xs
- IV. ∀ xs::[a] . ∀ f::(a->b) . length (map f xs) = length xs
- V. ∀ xs::[a] . ∀ p::a->Bool . ∀ e::a . ((elem e (filter p xs)) ⇒ (elem e xs)) (asumiendo Eq a)
- VI. ∀ xs::[a] . ∀ x::a . ponerAlFinal x xs = xs ++ (x:[])
- VII. reverse = foldr (\x rec -> rec ++ (x:[])) [] -> CASI
- VIII. ∀ xs::[a] . ∀ x::a . head (reverse (ponerAlFinal x xs)) = x

SUMA PEQUEÑA

3)

i) POR IND EST en XS

P(xs) = LENGTH (DUPLICAR xs) = 2 * LENGTH (xs)

CASO BASE:

P([ ]): LENGTH (DUPLICAR [ ]) = 2 * LENGTH ([ ])

- LENGTH (DUPLICAR [ ]) $\stackrel{\{D0\}}{=}$ LENGTH ([ ]) $\stackrel{\{L0\}}{=}$ 0
- 2 * LENGTH ([ ]) $\stackrel{\{L0\}}{=}$ 2 * 0 ...

2 * LENGTH([]) = 2 * 0 = 0

PASO INDUCTIVO:

- HI: LENGTH(DUPLICAR XS) = 2 * LENGTH(XS);
- TI: LENGTH(DUPLICAR(X:XS)) = 2 * LENGTH(X:XS);

LENGTH(DUPLICAR(X:XS))

$\overset{\{D1\}}{=}$ LENGTH($\underset{\sim}{\overset{x}{X}}$: X : $\overbrace{\text{DUPLICAR XS}}^{xs'}$)

$\overset{\{L1\}}{=}$ 1 + LENGTH(X : DUPLICAR XS)

$\overset{\{L1\}}{=}$ 2 + LENGTH(DUPLICAR XS)

$\overset{HI}{=}$ 2 + 2 * LENGTH (XS)

$\overset{*}{=}$ 2(1 + LENGTH(XS))

$\overset{\{L1\}}{=}$ 2 * LENGTH(X:XS) ✓ ✓

* FACTOR COMÚN

→ XS/YS en este caso da igual. Lo hago FOLD.

ii) P(XS) = ∀YS :: [a] . LENGTH (XS ++ YS) = LENGTH XS + LENGTH YS

CASO BASE: P([])

$\underbrace{\text{LENGTH([] ++ YS)}}_{1}$ = $\underbrace{\text{LENGTH [] + LENGTH YS}}_{2}$

1. LENGTH([] ++ YS) $\overset{\{++0\}}{=}$ LENGTH(YS)

POR LEMA GENERACIÓN DE LISTAS:

1. YS = []
2. ∃Z :: a . ∃ZS [a] . YS = (Z : ZS)

CASO 1: YS = []

LENGTH([]) $\overset{\{L0\}}{=}$ 0

CASO 2: YS = (Z : ZS)

LENGTH(Z : ZS) $\overset{\{L1\}}{=}$ 1 + LENGTH(ZS)

¿Tengo que hacerlo?

Si la respuesta es no, ¿cuándo?

2. LENGTH [] + LENGTH YS

$\overset{\{L0\}}{=}$ 0 + LENGTH YS

= LENGTH(YS) ✓

Se cumple el caso base.

P(XS) → P(X:XS)

Paso inductivo: Unir los ∀ pero se asume que están

- HI: $\text{length}(xs ++ ys) = \text{length } xs + \text{length } ys$
- TI: $\text{length}((x:xs) ++ ys) = \text{length}(x:xs) + \text{length } ys$

$$
\begin{aligned}
& \text{length}((x:xs) ++ ys) \\
& \overset{\{++1\}}{=} \text{length}(x:(xs ++ ys)) \\
& \overset{\{L1\}}{=} 1 + \text{length}(xs ++ ys) \\
& \overset{HI}{=} \underline{1 + \text{length } xs} + \text{length } ys \\
& \overset{\{L1\}}{=} \text{length}(x:xs) + \text{length } ys \quad \checkmark
\end{aligned}
$$

Como queríamos probar.

iii) $P(xs) = \forall x :: a \,.\, \text{append } [x] \; xs = x : xs$

Caso base: $P([\,])$

$$\underbrace{\text{append } [x] \; [\,]}_{1} = \underbrace{x : [\,]}_{2}$$

$\rightsquigarrow$ xs ++ ys   xs... (:) ys

1. $\text{append}[x][\,]$
$$
\begin{aligned}
& \overset{\{A0\}}{=} \text{foldr}(:) \, [\,] \, [x] \\
& \overset{\{F1\}}{=} x : (\text{foldr}(:) [\,] [\,]) \\
& \overset{\{F0\}}{=} x : [\,] \\
& \overset{\{:\}}{=} [x]
\end{aligned}
$$

2. $x : [\,]$
$$\overset{\{:\}}{=} [x]$$

Paso inductivo: $P(xs) \Rightarrow P(x:xs)$

- HI: $\text{append } [x] \; xs = x : xs$
- TI: $\text{append } [x] \; (x:xs) = x : (x:xs)$

$$
\begin{aligned}
& \text{append } [x] \; (x:xs) \\
& \overset{\{A0\}}{=} \text{foldr}(:) \; (x:xs) \; [x]
\end{aligned}
$$

{F1}

$=$ x : (FOLDR (:) (x:xs) [])

$\overset{\{FO\}}{=}$ x : (x:xs) ¿no apliqué HI?

depende de "MODIF" igual mismo cant elem.

iv) $P(xs) = (\forall f :: (a \to b))(\text{LENGTH}(\text{MAP } f\ xs) = \text{LENGTH } xs)$

CASO BASE: P([])

{M} MAP f l = FOLDR (\x rec -> f x : rec) [] l

$\underbrace{\text{LENGTH}(\text{MAP } f\ [])}_{1} = \underbrace{\text{LENGTH}([])}_{2}$

1. LENGTH(MAP f [])

$\overset{\{M\}}{=}$ LENGTH(FOLDR(\x rec -> f x : rec) [] [])

$\overset{\{F0\}}{=}$ LENGTH([])

$\overset{\{L0\}}{=}$ 0

2. LENGTH([])

$\overset{\{L0\}}{=}$ 0

PASO INDUCTIVO: $(\forall f : a \to b)(\forall xs : [a])(\forall x : a)(P(x) \Rightarrow P(x:xs))$

- HI: $(\forall f : a \to b)(\forall xs : [a])(\text{LENGTH}(\text{MAP } f\ xs) = \text{LENGTH}(xs))$
- TI: $(\forall f : a \to b)(\forall xs : [a])(\forall x : a)(\text{LENGTH}(\text{MAP } f\ (x:xs)) = \text{LENGTH}(x:xs))$

LENGTH(MAP f (x:xs))

$\overset{\{M\}}{=}$ LENGTH(FOLDR(...) [] (x:xs))

$\overset{\{F1\}}{=}$ LENGTH(x : (FOLDR(...) [] xs))

$\overset{\{M\}}{=}$ LENGTH(x : MAP f xs)

$\overset{\{L1\}}{=}$ 1 + LENGTH(MAP f xs)

$\overset{HI}{=}$ 1 + LENGTH xs

$\overset{\{L1\}}{=}$ LENGTH(x:xs) ✓

V) $P(xs) = (\forall xs : [a])(\forall p :: a \to \text{Bool})(\forall e :: a)((\text{ELEM } e\ (\text{FILTER } p\ xs)) \Rightarrow (\text{ELEM } e\ xs))$

Asumo antecedente V, caso contrario es trivial.

CASO BASE: P([])

$\underbrace{\text{ELEM } e\ (\text{FILTER } p\ [])}_{1} \Rightarrow \underbrace{\text{ELEM } e\ []}_{2}$

1. ELEM e (FILTER P [])

$\overset{\{F\}}{=}$ ELEM e (FOLDR (\x rec -> if P x then x:rec else rec) [] [])

$\overset{\{F0\}}{=}$ ELEM e []

$\overset{\{E0\}}{=}$ FALSE

2. ELEM e []

$\overset{\{E0\}}{=}$ FALSE

PASO INDUCTIVO: $(\forall xs::[a])(\forall x::a)(P(xs) \Rightarrow P(x:xs))$

$P(xs) : (\forall xs:[a])(\forall P:a\to Bool)(\forall e:a)(\text{ELEM } e \ (\text{FILTER } P \ xs) \Rightarrow \text{ELEM } e \ xs)$

$P(x:xs) : (\forall x:a)(\forall xs:[a])(\forall P:a\to Bool)(\forall e:a)(\text{ELEM } e \ (\text{FILTER } P \ (x:xs)) \Rightarrow \text{ELEM } e \ (x:xs))$

Asumo que los $\forall$ están siempre presentes y que el ANTECEDENTE es VERDADERO.

Asumo TRUE = ELEM e (FILTER P (x:xs)) ✓

e ESTÁ EN (x:xs) luego de filtrar con P.

ELEM e (x:xs)

$\overset{\{E1\}}{=}$ e == x || ELEM e xs

Por el lema de generación de booleanos

1) e == x = TRUE

2) e == x = FALSE

CASO (e == x = TRUE)

= TRUE || ELEM e xs

$\overset{\{Bool\}}{=}$ TRUE

```
   foldr :: (a -> (b -> b)) -> b -> [a] -> b
{F0}foldr f z [] = <
{F1}foldr f z (x:xs) = f x (foldr f z xs)
```

$vi)\ P(xs), \forall xs :: [a]$ tq $P(xs) \equiv (\forall x :: a)(\text{ponerAlFinal } x\ xs = xs \mathbin{++} (x:[\,]))$

Caso base: $P([\,])$

$$\underbrace{\text{ponerAlFinal } x\ [\,]}_{1} = \underbrace{[\,] \mathbin{++} (x:[\,])}_{2}$$

1. $\text{ponerAlFinal } x\ [\,]$

$\overset{\{P_0\}}{=} \text{foldr } (:)\ (x:[\,])\ [\,]$

$\overset{\{F_0\}}{=} (x:[\,])$

$\overset{\{:\}}{=} [x]$ ✓

2. $[\,] \mathbin{++} (x:[\,])$

$\overset{\{{+}{+}0\}}{=} x:[\,]$

$\overset{(:)}{=} [x]$ ✓

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

- $P(xs) = (\forall x :: a)(\text{ponerAlFinal } x\ xs = xs \mathbin{++} (x:[\,]))$
- $P(x':xs) = (\forall x' :: a)(\forall x :: a)(\text{poner-alFinal } x\ (x':xs) = (x':xs) \mathbin{++} (x:[\,]))$

$\text{ponerAlFinal } x\ (x':xs)$

$\overset{\{P_0\}}{=} \text{foldr}(:)\ (x:[\,])\ (x':xs)$

$\overset{\{:\}}{=} \text{foldr } (:)(x:[\,])(x':xs)$

$\overset{\{F_1\}}{=} (:)\, x'\ (\text{foldr } (:)\ (x:[\,])\, xs)$

$\overset{\{P_0\}}{=} (:)\, x'\ (\text{ponerAlFinal } x\ xs)$

$\overset{\{HI\}}{=} (:)\, x'\ (xs \mathbin{++} (x:[\,]))$

$\overset{\{:\}}{=} x' : (xs \mathbin{++} (x:[\,]))$

$\overset{\{{+}{+}1\}}{=} (x':xs) \mathbin{++} [x]$ que era a lo que queríamos llegar

No haría hacer este lema pues hay impl de ++ que lo hace (en clase).

$(x':xs) \mathbin{++} (x:[\,]))$

$1:([2,3] \mathbin{++} [4]) = 1:([2,3,4]) = [1,2,3,4]$

$(1:[2,3]) \mathbin{++} (4:[\,])) = [1,2,3] \mathbin{++} [4] = [1,2,3,4]$

vii) Por el pp.io de ext funcional, $(\forall xs::[a])(\text{reverse } xs = \text{foldr } (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ []\ xs)$

Por ind estructural en xs:

$P(xs): (\forall xs::[a])(\text{reverse } xs = \text{foldr } (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ []\ xs)$

$P(x':xs): (\forall xs::[a])(\forall x'::a)(\text{reverse } (x':xs) = \text{foldr } (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ []\ (x':xs))$

$\text{reverse } (x':xs)$

$\{R_0\}$

$= \text{foldl } (\text{flip } (:))\ []\ (x':xs)$

$\{F_{L1}\}$

$= \text{foldl } (\text{flip } (:))\ (\text{flip } (:)\ []\ x')\ xs$

$\{\text{flip}\}$

$= \text{foldl } (\text{flip } (:))\ ((:)\ []\ x')\ xs$

$\{:\}$

* $= \text{foldl } (\text{flip } (:))\ [x']\ xs$

$= \text{foldr } (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ []\ (x':xs)$

$\{F_1\}$

$= (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ x'\ (\text{foldr } (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ []\ xs)$

$\{\beta\}$

$= (\backslash rec \to rec \mathbin{++} (x':[]))\ (\text{foldr } (\backslash x\ rec \to rec \mathbin{++} (x:[]))\ []\ xs)$

$\{HI\}$

$= (\backslash rec \to rec \mathbin{++} (x':[]))\ \text{reverse } xs$

$\{\beta\}$

* $= \text{reverse } xs \mathbin{++} (x':[])$

$\{:\}$

$= \text{reverse } xs \mathbin{++} ([x'])$

$\text{foldl } (\text{flip } (:))\ [x']\ xs$

$\text{foldl } (\text{flip } (:))\ [x']\ xs \overset{\{R_0\}}{=} (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} [x']$

rec mas externa.

Lema: $\text{foldl } (\text{flip } (:))\ ys\ xs = (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} ys$

$P(ys): (\forall xs::[a])(\forall ys::[a])(\text{foldl } (\text{flip } (:))\ ys\ xs = (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} ys)$

Caso base: $P([])$

$\text{foldl } (\text{flip } (:))\ []\ xs = (\text{foldl } (\text{flip } (:))\ []\ xs)) \mathbin{++} []$

1. $\text{foldl } (\text{flip } (:))\ []\ xs$ ✓ $= (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} []$

$\{{++}_0\}$ *

$= \text{foldl } (\text{flip } (:))\ []\ xs$ ✓

* Pruebo luego $xs \mathbin{++} [] = xs$

Paso inductivo: $(\forall ys::[a])(\forall y::a)(P(ys) \Rightarrow P(y:ys))$

- $P(ys): (\forall xs::[a])(\forall ys::[a])(\text{foldl } (\text{flip } (:))\ ys\ xs = (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} ys)$
- $P(y:ys): (\forall xs::[a], y::a, ys::[a])(\text{foldl } (\text{flip } (:))\ (y:ys)\ xs = (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} (y:ys))$

$\text{foldl } (\text{flip } (:))\ (y:ys)\ xs = (\text{foldl } (\text{flip } (:))\ []\ xs) \mathbin{++} (y:ys)$

Por lema de generación de listas tengo dos casos:

1. $xs = []$

2. $(\exists z::a, zs::[a])(xs = (z:zs))$

Caso 1:

```
foldl :: (b -> a -> b) -> b -> [a] -> b
{Fl0} foldl f ac [] = ac
{Fl1} foldl f ac (x:xs) = foldl f (f ac x) xs
```

foldl (flip(:)) (y:ys) [] = (foldl (flip(:)) [] []) ++ (y:ys)

{Fl0}
⟺ (y:ys) = (foldl (flip(:)) [] []) ++ (y:ys)

{Fl0}
⟺ (y:ys) = [] ++ (y:ys)

{++0}
⟺ (y:ys) = (y:ys) ✓

CASO 2:

foldl (flip(:)) $\underbrace{(y:ys)}_{xs}$ (z:zs) = (foldl (flip(:)) [] $\underbrace{(z:zs)}_{xs}$) ++ (y:ys)

{Fl1}
⟺ foldl (flip(:)) ((flip(:)) (y:ys) z) zs = (foldl (flip(:)) [] (z:zs)) ++ (y:ys)

{Fl1}
⟺ foldl (flip(:)) ((flip(:)) (y:ys) z) zs = (foldl (flip(:)) ((flip(:)) [] z) zs) ++ (y:ys)

{flip(:)}
⟺ foldl (flip(:)) ((y:ys) : z) zs = (foldl (flip(:)) ([] : z) zs) ++ (y:ys)

{:}
⟺ foldl (flip(:)) ((y:ys) : z) zs = (foldl (flip(:)) [z] zs) ++ (y:ys)

[a] -> a -> [a]

= Sigo dsp.

Ejercicio 5 ★

Dadas las siguientes funciones:

```
      zip :: [a] -> [b] -> [(a,b)]
{Z0}  zip = (foldr (\x rec ys ->
         xs ys      if null ys
                      then []
                      else (x, head ys) : rec (tail ys))
                  (const []) xs) ys

      zip' :: [a] -> [b] -> [(a,b)]
{Z'0} zip' []     ys = []
{Z'1} zip' (x:xs) ys = if null ys then [] else (x, head ys):zip' xs (tail ys)
```

Demostrar que zip = zip' utilizando inducción estructural y el principio de extensionalidad.

Antes de una inducción, necesitamos usar el ppio. de ext funcional para demostrar que vale la igualdad para xs, ys.

Por el ppio. de ext. funcional: (∀xs::[a])(∀ys::[b])(zip xs ys = zip' xs ys)

?

Ahora, por INDUCCIÓN: P(xs) = (∀ys::[b])(zip xs ys = zip' xs ys)

CASO BASE: P([]): $\underbrace{\text{zip [] ys}}_{1.}$ = $\underbrace{\text{zip' [] ys}}_{2.}$

1. zip [] ys
{Z0}
= (foldr (...) (const []) []) ys
{F0}
= (const []) ys
{C}
= []

foldr f z [] = z

foldr f z (x:xs) = f x (foldr f z xs)

const x y = x

2. zip' [] ys
{Z0'}
= []

Se cumple el caso base :)

PASO INDUCTIVO: $P(xs) \Rightarrow P(x:xs)$

· $P(xs): (\forall ys :: [b])(\text{zip } xs\ ys = \text{zip}'\ xs\ ys)$

· $P(x:xs): (\forall ys :: [b])(\underbrace{\text{zip } (x:xs)\ ys}_{1} = \underbrace{\text{zip}'\ (x:xs)\ ys}_{2})$

Asumo que los $\forall$ están siempre presentes.

? Lo entiendo BIEN cuando el ys entra en ACCIÓN. ¿Después de haber TERMINADO el FOLD?

1. zip (x:xs) ys
{Z0}
= (foldr (\x rec ys -> if null ys then [] else (x, head ys) : rec (tail ys)) (const []) (x:xs)) ys
{F1}
= ((\x rec ys -> if null ys then [] else (x, head ys) : rec (tail ys)) x (foldr f (const []) xs)) ys
{B}
= ((\rec ys -> if null ys then [] else (x, head ys) : rec (tail ys))
(foldr (\x rec ys -> if null ys then [] else (x, head ys) : rec (tail ys)) (const []) xs)) ys
zip xs
= ((\rec ys -> if null ys then [] else (x, head ys) : rec (tail ys)) (zip xs)) ys
{B}
= (\ys -> if null ys then [] else (x, head ys) : zip xs (tail ys)) ys
{B}
= if null ys then [] else (x, head ys) : zip xs (tail ys)
Hi, xq vale $\forall$ ys.
{Hi}
= if null ys then [] else (x, head ys) : zip' xs (tail ys)

Por el lema de generación de listas en ys

1. ys = []

2. $(\exists z :: b, zs :: [b])(ys = (z:zs))$

Caso 1 (ys = [])

if null [] then [] else (x, head []) : zip' xs (tail [])
{IfT}
= [] ✓

zip' (x:xs) []

{Z'1}
= if null [] then [] else (x, head []) : zip' xs (tail [])
{if T}
= [] ✓

Caso 2 | ys : (z:zs)

if null (z:zs) then [] else (x, head (z:zs)) : zip' xs (tail (z:zs))
{if F}
= (x, head (z:zs)) : zip xs (tail (z:zs))
{H}
= (x, z) : zip xs (tail (z:zs))
{T}
= (x, z) : zip xs zs. V por HI.

Ejercicio 5 ★

Dadas las siguientes funciones:

```
      zip :: [a] -> [b] -> [(a,b)]
{Z0}  zip = foldr (\x rec ys ->
                    if null ys
                      then []
                      else (x, head ys) : rec (tail ys))
                  (const [])

      zip' :: [a] -> [b] -> [(a,b)]
{Z'0} zip' []     ys = []
{Z'1} zip' (x:xs) ys = if null ys then [] else (x, head ys):zip' xs (tail ys)
```

Demostrar que zip = zip' utilizando inducción estructural y el principio de extensionalidad.

zip' (x:xs) (z:zs) = if null (z:zs) then [] else (x, head (z:zs)) : zip' xs (tail (z:zs))
{if F}
= (x, head (z:zs)) : zip' xs (tail (z:zs))
{H}
= (x, z) : zip' xs (tail (z:zs))
{T}
= (x, z) : zip' xs zs. V por HI.

Ejercicio 6 ★

Dadas las siguientes funciones:

```
     nub :: Eq a => [a] -> [a]
{N0} nub [] = []
{N1} nub (x:xs) = x : filter (\y -> x /= y) (nub xs)

     union :: Eq a => [a] -> [a] -> [a]
{U0} union xs ys = nub (xs++ys)

     intersect :: Eq a => [a] -> [a] -> [a]
{I0} intersect xs ys = filter (\e -> elem e ys) xs
```

Indicar si las siguientes propiedades son verdaderas o falsas. Si son verdaderas, realizar una demostración. Si son falsas, presentar un contraejemplo.

- I. Eq a => ∀ xs::[a] . ∀ e::a . ∀ p::a -> Bool . elem e xs && p e = elem e (filter p xs)
- II. Eq a => ∀ xs::[a] . ∀ e::a . elem e xs = elem e (nub xs)
- III. Eq a => ∀ xs::[a] . ∀ ys::[a] . ∀ e::a . elem e (union xs ys) = (elem e xs) || (elem e ys)
- IV. Eq a => ∀ xs::[a] . ∀ ys::[a] . ∀ e::a . elem e (intersect xs ys) = (elem e xs) && (elem e ys)
- V. Eq a => ∀ xs::[a] . ∀ ys::[a] . length (union xs ys) = length xs + length ys
- VI. Eq a => ∀ xs::[a] . ∀ ys::[a] . length (union xs ys) ≤ length xs + length ys

i) VERDADERA.

Elem e xs && p e = e está en xs, e cumple filtro p.

Elem e (filter p xs) = filtra elems xs usando p. Luego elem e xs.

Asunción: asumo ant V. De lo contrario trivial V ✓

P(xs): (Eq a) => (∀e::a) (∀p::a) (elem e xs && p e = elem e (filter p xs))

Asumo que los ∀ siempre están presentes.

Caso base: P([])

$\underbrace{\text{elem e [] \&\& p e}}_{1} = \underbrace{\text{elem e (filter p [])}}_{2}$

1. elem e [] && p e
{E0}
= False ✓

2. elem e (filter p [])
{F}
= elem e (foldr (\x rec -> if p x then x : rec else rec) [] [])
{F00}

$= Elem\ e\ []$

$\overset{\{E0\}}{=}$ FALSE ✓

Luego, el caso P([]) vale ✓

PASO INDUCTIVO: $(\forall xs::[a])(\forall x::a)(P(xs) \Rightarrow P(x:xs))$

- $P(xs)$: $(\forall e::a)(\forall p::(a \to Bool))$ (Elem e xs && P e = Elem e (Filter P xs))
- $P(x:xs)$: $(\forall e::a)(\forall p::(a \to Bool))$(Elem e (x:xs) && P e = Elem e (Filter P (x:xs)))

=>)

(Elem e (x:xs)) && P e

$\overset{\{E1\}}{=}$ (e == x || Elem e xs) && P e *

Por lema de generación de booleanos tengo dos casos *

1. e == x = TRUE
2. e == x = FALSE

Caso 1 (e == x = TRUE)

* (TRUE || Elem e xs) && P e

$\overset{Bool}{=}$ TRUE && P e = P e

Por lema de generación de booleanos tengo dos casos.

1.a: P e == TRUE

1.b : P e == FALSE

Caso 1.a: (e == x = TRUE && P e == TRUE)

TRUE && TRUE = TRUE ✓

Caso 1.b: (e == x = TRUE && P e == FALSE)

TRUE && FALSE = FALSE ✓

{F0} Filter P [] = []

{F1} Filter P (x:xs) = if P x then x : Filter P xs else Filter P xs

Caso 2 (e == x = FALSE)

* (FALSE || Elem e xs) && P e

$\overset{Bool}{=}$ Elem e xs && P e

$\overset{\{HI\}}{=}$ Elem e (Filter P xs) True por HI

{E0} Elem e [] = FALSE

{E1} Elem e (x:xs) = (e == x) || Elem e xs

LD: Caso 1 (e == x = T, P e = T o P e = F)

Elem e (Filter P (x:xs))

$\overset{\{F1\}}{=}$ ELEM e (IF Px THEN x:FILTER P xs
ELSE FILTER P xs)

POR LEMA DE GENERACIÓN DE BOOLEANOS.

1. Px = T

2. Px = F

1. ELEM e (if TRUE ...)

e == x = T

$\overset{\{IFT\}}{=}$ ELEM e (x:FILTER P xs)

{NIP}

$\overset{\{E1\}}{=}$ (e == x) || ELEM e (FILTER P xs) = TRUE ✓

2. ELEM e (IF FALSE THEN x:FILTER P xs ELSE FILTER P xs)

$\overset{\{IFF\}}{=}$ ELEM e (FILTER P xs) ✓ TRUE POR HI

LD: CASO 2 (e == x = F)

ELEM e (FILTER P (x:xs))

ELEM e (IF Px THEN x:FILTER P xs ELSE FILTER P xs)

POR EL LEMA DE GENERACIÓN DE BOOL

1. Px = T

2. Px = F

1. ELEM e (IF TRUE ...)

$\overset{\{IFT\}}{=}$ ELEM e (x:FILTER P xs)

$\overset{\{E1\}}{=}$ e == x || ELEM e (FILTER P xs)

$\overset{\{NIP\}}{=}$ FALSE || ELEM e (FILTER P xs)

$\overset{\{Bool\}}{=}$ ELEM e (FILTER P xs) TRUE POR HI

2. ELEM e (IF FALSE THEN x:FILTER P xs ELSE FILTER P xs)

$\overset{\{IFF\}}{=}$ ELEM e (FILTER P xs) TRUE POR HI

ii) P(xs) = EQ D- => (∀xs::[a])(∀e::a)(ELEM e xs = ELEM e (NUB xs))

CASO BASE: P([])

ELEM e [] = ELEM e (NUB [])

$\overset{\{NO\}}{\Leftrightarrow}$ ELEM e [] = ELEM e []

Luego, el caso base es verdadero.

{F1} FILTER P (x:xs) = IF Px
THEN x:FILTER P xs
ELSE FILTER P xs

PASO INDUCTIVO: (∀xs :: [a]) (∀x :: a) (P(xs) => P(x:xs))

```
P(xs): EQ a => (∀e::a) (ELEM e xs = ELEM e (NUB xs))

P(x:xs): EQ a => (∀x::a) (∀e::a) (ELEM e (x:xs) = ELEM e (NUB (x:xs)))
```

```
ELEM e (x:xs) = ELEM e (NUB (x:xs))
{E0}
<=> FOLDR (\x -> (||) (x==e)) FALSE (x:xs) = ELEM e (NUB (x:xs))
{N1}
<=> FOLDR (\x -> (||) (x==e)) FALSE (x:xs) = ELEM e (x: FILTER (\y -> x ≠ y) (NUB xs))
{F1}
<=> (\x -> (||) (x==e)) x (FOLDR (\x -> (||) x==e) FALSE xs) = ELEM e (x: FILTER (\y -> x≠y) (NUB xs))
{B}
<=> ((||) x==e) (FOLDR (\x -> (||) x==e) FALSE xs) = ELEM e (x: FILTER (\y -> x ≠ y) (NUB xs))
{E0}
<=> (||) (x==e) (ELEM e xs) = ELEM e (x: FILTER (\y -> y ≠ x) (NUB xs))
{HI}
<=> (||) (x==e) (ELEM e (NUB xs)) = ELEM e (x: FILTER (\y -> y ≠ x) (NUB xs))
{E0}
= (||) (x==e) (ELEM e (NUB xs)) = FOLDR (\x -> (||) (x==e)) FALSE (x: FILTER (\y -> y ≠ x) (NUB xs))
{F1}
= (||) (x==e) (ELEM e (NUB xs)) = (\x -> (||) (x==e)) x (FOLDR (\x -> (||) (x==e)) FALSE (FILTER (\y -> y ≠ x) (NUB xs)))
{B}
= (||) (x==e) (ELEM e (NUB xs)) = (||) (x==e) (FOLDR (\x -> (||) (x==e)) FALSE (FILTER (\y -> y ≠ x) (NUB xs)))
{E0}
= (||) (x==e) (ELEM e (NUB xs)) = (||) (x==e) (ELEM e (FILTER (\y -> y ≠ x) (NUB xs)))
{EJ ANT}
= (||) (x==e) (ELEM e (NUB xs)) = (||) (x==e) (ELEM e (NUB xs) && (\y -> y ≠ x) e)
{B}
= (||) (x==e) (ELEM e (NUB xs)) = (||) (x==e) (ELEM e (NUB xs) && e ≠ x)
```

Por lema de generación de booleanos tenemos dos casos: (OBS, e==x EVALÚA e≠x TRUE -> !(e==x))

1. e==x : TRUE

2. e==x: FALSE

CASO 1:

```
<=> (||) (x==e) (ELEM e (NUB xs)) = (||) (x==e) (ELEM e (NUB xs) && FALSE)

<=> (||) TRUE (ELEM e (NUB xs)) = (||) TRUE (ELEM e (NUB xs) && FALSE)
{BOOL, LAZY}
<=> TRUE = TRUE ✓
```

CASO 2:

```
<=> (||) (x==e) (ELEM e (NUB xs)) = (||) (x==e) (ELEM e (NUB xs) && !(e==x))

<=> (||) FALSE (ELEM e (NUB xs)) = (||) FALSE (ELEM e (NUB xs) && ! FALSE)
{BOOL}
<=> ELEM e (NUB xs) = ELEM e (NUB xs) && TRUE
{BOOL}
<=> ELEM e (NUB xs) = ELEM e (NUB xs) ✓
```

```
ELEM :: EQ a => a -> [a] -> Bool

ELEM e = FOLDR (\x -> (||) (x==e)) FALSE {E0}

FILTER :: (a -> Bool) -> [a] -> [a]

FILTER p = FOLDR (\x z -> if p x THEN x:z ELSE z) [] {F0}

FOLDR :: (a -> b -> b) -> b -> [a] -> b

FOLDR f z [] = z

FOLDR f z (x:xs) = f x (FOLDR f z xs)
```

3) P(…) EQ a => (∀ …

$P(xs)$: EQ a => $(\forall xs:[a])(\forall ys:[a])(\forall e:a)$ (Elem e (union xs ys) = (Elem e xs) || (Elem e ys))

Caso base: P([])

Elem e (union [] ys) = (Elem e []) || (Elem e ys)
{U0}
<=> Elem e (nub([]++ys)) = (Elem e []) || (Elem e ys)
{++0}
<=> Elem e (nub ys) = (Elem e []) || (Elem e ys)
{Ej ant}
<=> Elem e ys = (Elem e []) || (Elem e ys)
{E0}
<=> Elem e ys = (foldr (\x -> (||) (x==e)) False []) || (Elem e ys)
{F0}
<=> Elem e ys = False || Elem e ys
{Bool}
<=> Elem e ys = Elem e ys ✓

se cumple el caso base.

Paso inductivo: EQ a => $(\forall xs:[a])(\forall x::a)$ (P(xs) => P(x:xs))

P(xs): EQ a => $(\forall ys:[a])(\forall e:a)$(Elem e (union xs ys) = (Elem e xs) || (Elem e ys))

P(x:xs): EQ a => $(\forall ys:[a])(\forall e:a)$(Elem e (union (x:xs) ys) = (Elem e (x:xs)) || (Elem e ys))

Elem e (union (x:xs) ys) = (Elem e (x:xs)) || (Elem e ys)
{U0}
<=> Elem e (nub((x:xs)++ys)) = (Elem e (x:xs)) || (Elem e ys)
{++1}
<=> Elem e (nub(x:(xs++ys))) = (Elem e (x:xs)) || (Elem e ys)
{N1}
<=> Elem e (x: filter (\y -> x≠y) (nub(xs++ys))) = (Elem e (x:xs)) || (Elem e ys)
{E0}
<=> foldr (\x -> (||) (x==e)) False (x: filter (\y -> x≠y) (nub(xs++ys))) = (Elem e (x:xs)) || (Elem e ys)
{F1}
<=> (\x -> (||) (x==e)) x (foldr (\x -> (||) (x==e)) False (filter (\y -> x≠y) (nub(xs++ys))) = (Elem e (x:xs)) || (Elem e ys)
{B}
<=> (||) (x==e) (foldr (\x -> (||) (x==e)) False (filter (\y -> x≠y) (nub (xs++ys))) = (Elem e (x:xs)) || (Elem e ys)
{E0}
<=> (||) (x==e) (Elem e (filter (\y -> x≠y) (nub (xs++ys))) = (Elem e (x:xs)) || (Elem e ys)
{Ej.Ant}
<=> (||) (x==e) (Elem e (nub (xs++ys)) && ((\y -> x≠y) e) = (Elem e (x:xs)) || (Elem e ys)
{U0}
<=> (||) (x==e) (Elem e (union xs ys) && ((\y -> x≠y) e) = (Elem e (x:xs)) || (Elem e ys)
{HI}
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && ((\y -> x≠y) e)) = (Elem e (x:xs)) || (Elem e ys)
{E0}
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && ((\y -> x≠y) e)) = (foldr (\x -> (||) (x==e)) False (x:xs)) || (Elem e ys)
{F1}
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && ((\y -> x≠y) e)) = (\x -> (||) (x==e)) x (foldr (\x -> (||) (x==e)) False xs) || Elem e ys
{B}
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && ((\y -> x≠y) e)) = ((||) (x==e) (foldr (\x -> (||) (x==e)) False xs)) || Elem e ys
E0
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && ((\y -> x≠y) e)) = ((||) (x==e) (Elem e xs)) || Elem e ys
Conm: {||}
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && ((\y -> x≠y) e)) = (||) (x==e) (Elem e xs || Elem e ys)
{B}
<=> (||) (x==e) (((Elem e xs) || (Elem e ys)) && x≠e)) = (||) (x==e) (Elem e xs || Elem e ys)

INFIJO {||}
<=> (x==e) || (((elem e xs) || (elem e ys)) && !(x==e)) = (x==e) || (elem e xs || elem e ys)

Por el lema de generación de booleanos tengo dos casos en (x==e)

1. x==e : TRUE

2. x==e : FALSE

Caso 1:

(x==e) || (((elem e xs) || (elem e ys)) && !(x==e)) = (x==e) || (elem e xs || elem e ys)

<=> TRUE || (((elem e xs) || (elem e ys)) && FALSE) = TRUE || (elem e xs || elem e ys)
{LAZY}
{Bool}
<=> TRUE = TRUE ✓

Caso 2:

(x==e) || (((elem e xs) || (elem e ys)) && !(x==e)) = (x==e) || (elem e xs || elem e ys)

<=> FALSE || (((elem e xs) || (elem e ys)) && TRUE) = FALSE || (elem e xs || elem e ys)
{Bool}
<=> ((elem e xs) || (elem e ys)) && TRUE = (elem e xs || elem e ys)
{Bool}
<=> elem e xs || elem e ys = elem e xs || elem e ys ✓

4) P(xs): Eq a => (∀xs::[a])(∀ys::[a])(∀e::a)(elem e (intersect xs ys) = (elem e xs) && (elem e ys))

Por inducción en xs.

Caso base: P([])

elem e (intersect [] ys) = (elem e []) && (elem e ys)
{I0}
<=> elem e (filter (\e -> elem e ys) []) = (elem e []) && (elem e ys)
{Fi0}
<=> elem e (foldr (\x z -> if (\e -> elem e ys) x then x:z else z) [] []) = (elem e []) && (elem e ys)
{Fo}
<=> elem e [] = elem e [] && elem e ys
{E0}
<=> foldr (\x -> (||) (x==e)) FALSE [] = foldr (\x -> (||) (x==e)) FALSE [] && foldr (\x -> (||) (x==e)) FALSE ys
{F0}
<=> FALSE = FALSE && foldr (\x -> (||) (x==e)) FALSE ys
{Bool}
<=> FALSE = FALSE ✓

Paso inductivo: (Eq a) => (∀xs::[a])(∀x::a)(P(xs) => P(x:xs))

P(xs): Eq a => (∀xs::[a])(∀ys::[a])(∀e::a)(elem e (intersect xs ys) = (elem e xs) && (elem e ys))

P(x:xs): Eq a => (∀xs::[a])(∀ys::[a])(∀e::a)(elem e (intersect (x:xs) ys) = (elem e (x:xs)) && (elem e ys))

elem e (intersect (x:xs) ys) = (elem e (x:xs)) && (elem e ys)
{I0}
<=> elem e (filter (\e -> elem e ys) (x:xs)) = (elem e (x:xs)) && (elem e ys)
{Fi0}
<=> elem e (foldr (\x z -> if (\e -> elem e ys) x then x:z else z) [] (x:xs)) = (elem e (x:xs)) && (elem e ys)
{F1}
<=>

<=> Elem e ((\x z -> if (\e -> Elem e ys) x THEN x:z ELSE z) x (Foldr ((\x z -> if (\e -> Elem e ys) x THEN x:z ELSE z) [] xs)) =

{B}
<=> Elem e ((\x z -> if Elem x ys THEN x:z ELSE z) x (Foldr ((\x z -> if (\e -> Elem e ys) x THEN x:z ELSE z) [] xs)) =

{B}
<=> Elem e ((\z -> if Elem x ys THEN x:z ELSE z) (Foldr ((\x z -> if (\e -> Elem e ys) x THEN x:z ELSE z) [] xs)) = (Elem e (x:xs)) && (Elem e ys)

{F:0}
<=> Elem e ((\z -> if Elem x ys THEN x:z ELSE z) (Filter (\e -> Elem e ys) xs)) = (Elem e (x:xs)) && (Elem e ys)

{I0}
<=> Elem e ((\z -> if Elem x ys THEN x:z ELSE z) (intersect xs ys)) = (Elem e (x:xs)) && (Elem e ys)

{B}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = (Elem e (x:xs)) && (Elem e ys)

{E0}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = (Foldr (\x -> (||) (x==e)) False (x:xs)) && (Elem e ys)

{F1}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = ((\x -> (||) (x==e)) x (Foldr (\x -> (||) (x==e)) False xs)) && (Elem e ys)

{B}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = ((||) (x==e) (Foldr (\x -> (||) (x==e)) False xs) && (Elem e ys)

{E0}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = ((||) (x==e) Elem e xs) && (Elem e ys)

{||}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = (x==e || Elem e xs) && (Elem e ys)

{Dist||}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = (x==e && Elem e ys) || (Elem e xs && Elem e ys)

{Hi}
<=> Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = (x==e && Elem e ys) || (Elem e (intersect xs ys))

Por el lema de generación de booleanos tengo dos casos:

1. Elem x ys = True

2. Elem x ys = False

Caso 1: Elem e (if Elem x ys THEN x:(intersect xs ys) ELSE (intersect xs ys)) = (x==e && Elem e ys) || (Elem e (intersect xs ys))

<=> Elem e (if True THEN x:(intersect xs ys) ELSE (intersect xs ys)) = ((x==e) && Elem e ys) || (Elem e (intersect xs ys))

{IF1}
<=> Elem e (x:(intersect xs ys)) = ((x==e) && Elem e ys) || (Elem e (intersect xs ys))

{Elem}
<=> (e==x) || Elem e (intersect xs ys) = ((x==e) && Elem e ys) || (Elem e (intersect xs ys))

Por el lema de generación de booleanos tengo dos casos:

a. (x==e) = True

b. (x==e) = False

Caso 1.a: (Elem x ys = True && (x==e = True))

{||}
<=> (x==e) || (Elem e (intersect xs ys)) = ((x==e) && Elem e ys) || (Elem e (intersect xs ys))

{Bool}
<=> True || (Elem e (intersect xs ys)) = (True && Elem e ys) || (Elem e (intersect xs ys))

Elem e ys = Elem x ys

{x==e}
<=> True = Elem x ys || (Elem e (intersect xs ys))

Elem x ys = True

{Hip}
<=> True = True ✓

Caso 1.b (Elem x ys = True && (x==e = False))

{||}
<=> (x==e) || (Elem e (intersect xs ys)) = ((x==e) && Elem e ys) || (Elem e (intersect xs ys))

<=> False || (Elem e (intersect xs ys)) = (False && Elem e ys) || (Elem e (intersect xs ys))

{bool}
$\iff$ ELEM e (INTERSECT xs ys) = ELEM e (INTERSECT xs ys) ✓

Caso 2: ELEM x ys = FALSE

$\iff$ ELEM e (if ELEM x ys THEN x:(INTERSECT xs ys) ELSE (INTERSECT xs ys)) = (x==e && ELEM e ys) || (ELEM e (INTERSECT xs ys))

$\iff$ ELEM e (if FALSE THEN x:(INTERSECT xs ys) ELSE (INTERSECT xs ys)) = (x==e && ELEM e ys) || (ELEM e (INTERSECT xs ys))

{ifF}
$\iff$ ELEM e (INTERSECT xs ys) = ((x==e) && ELEM e ys) || (ELEM e (INTERSECT xs ys))

Por el lema de generación de booleanos tengo dos casos

a. (e==x) : TRUE

b. (e==x) : FALSE

Caso 2.a: (ELEM x ys = FALSE && (e==x = TRUE))

$\iff$ ELEM e (INTERSECT xs ys) = ((x==e) && ELEM e ys) || (ELEM e (INTERSECT xs ys))

$\iff$ ELEM e (INTERSECT xs ys) = (TRUE && ELEM e ys) || (ELEM e (INTERSECT xs ys))

{bool}
$\iff$ ELEM e (INTERSECT xs ys) = ELEM e ys || (ELEM e (INTERSECT xs ys))

{e:=x}
$\iff$ ELEM e (INTERSECT xs ys) = ELEM x ys || (ELEM e (INTERSECT xs ys))

{HIP}
$\iff$ ELEM e (INTERSECT xs ys) = FALSE || (ELEM e (INTERSECT xs ys)) ->FALSE

{bool}
$\iff$ ELEM e (INTERSECT xs ys) = ELEM e (INTERSECT xs ys) ✓

CASO 2.b (ELEM x ys = FALSE, (e==x = FALSE))

$\iff$ ELEM e (INTERSECT xs ys) = ((x==e) && ELEM e ys) || (ELEM e (INTERSECT xs ys))

$\iff$ ELEM e (INTERSECT xs ys) = (FALSE && ELEM e ys) || (ELEM e (INTERSECT xs ys))

{Bool}
$\iff$ ELEM e (INTERSECT xs ys) = (ELEM e (INTERSECT xs ys)) ✓

Ejercicio 9 ★

Dadas las funciones `altura` y `cantNodos` definidas en la práctica 1 para árboles binarios, demostrar la siguiente propiedad:

$\forall$ `x::AB a . altura x` $\leq$ `cantNodos x`

P(x) = ($\forall$x:AB a)(ALTURA x $\leq$ CANTNODOS x)

CASO BASE: P(NIL)

$\iff$ ALTURA NIL $\leq$ CANTNODOS NIL

{A}
$\iff$ FOLDAB (\lc x rc -> 1+ MAX lc rc) 0 NIL $\leq$ CANTNODOS NIL

{FAB0}
$\iff$ 0 $\leq$ CANTNODOS NIL

{CN}
$\iff$ 0 $\leq$ FOLDAB (\lc x rc -> 1+lc+rc) 0 NIL

{FAB0}
$\iff$ 0 $\leq$ 0

PASO INDUCTIVO: ($\forall$i:AB a)($\forall$d:AB a)($\forall$r:a)(P(i) $\wedge$ P(d) => P(Bin i r d))

· P(i): ($\forall$i:AB a)(ALTURA i $\leq$ CANTNODOS i)

P(d): (∀d:AB a)(Altura d ≤ CantNodos d)

·P(Bin i r d): (∀r:a)(∀i:AB a)(∀d:AB a)(Altura (Bin i r d) ≤ CantNodos (Bin i r d))

Altura (Bin i r d) ≤ CantNodos (Bin i r d)
{A}
<=> FoldAB (\lc x rc -> 1 + max lc rc) 0 (Bin i r d) ≤ CantNodos (Bin i r d)
{CN}
<=> FoldAB (\lc x rc -> 1 + max lc rc) 0 (Bin i r d) ≤ FoldAB (\lc x rc -> 1 + lc + rc) 0 (Bin i r d)
{FAB1}
<=> (\lc x rc -> 1+max lc rc) (FoldAB (\lc x rc -> 1+max lc rc) 0 i) r (FoldAB (\lc x rc -> 1+max lc rc) 0 d) ≤
{A}
<=> (\lc x rc -> 1+max lc rc) (Altura i) r (Altura d) ≤ FoldAB (\lc x rc -> 1+ lc+rc) 0 (Bin i r d)
{HI}
<=> (\lc x rc -> 1+max lc rc) (CantNodos i) r (CantNodos d) ≤ FoldAB (\lc x rc -> 1+ lc+rc) 0 (Bin i r d)
{FAB1}
<=> (\lc x rc -> 1+max lc rc) (CantNodos i) r (CantNodos d) ≤ (\lc x rc -> 1+lc+rc) (FoldAB (\lc x rc -> 1+lc+rc) 0 i) r (FoldAB (\lc x rc -> 1+lc+rc) 0 d)
{CN}
<=> (\lc x rc -> 1+max lc rc) (CantNodos i) r (CantNodos d) ≤ (\lc x rc -> 1+lc+rc) (CantNodos i) r (CantNodos d)
{B}
<=> (\x rc -> 1+max (CantNodos i) rc) r (CantNodos d) ≤ (\x rc -> 1+(CantNodos i)+rc) r (CantNodos d)
{B}
<=> (\rc -> 1+max (CantNodos i) rc) (CantNodos d) ≤ (\rc -> 1+(CantNodos i)+rc) (CantNodos d)
{B}
<=> 1+max (CantNodos i) (CantNodos d) ≤ 1+(CantNodos i) + (CantNodos d)
{max}
<=> 1+(if (CantNodos i) ≥ (CantNodos d) then (CantNodos i) else (CantNodos d)) ≤ 1+(CantNodos i) + (CantNodos d)

Por el lema de generación de Bool tengo dos casos:

1. (CantNodos i) ≥ (CantNodos d) = True

2. (CantNodos i) ≥ (CantNodos d) = False

Caso 1:

<=> 1+(if (CantNodos i) ≥ (CantNodos d) then (CantNodos i) else (CantNodos d)) ≤ 1+(CantNodos i) + (CantNodos d)

<=> 1+(if True then (CantNodos i) else (CantNodos d)) ≤ 1+(CantNodos i) + (CantNodos d)
{IfT}
<=> 1+ CantNodos(i) ≤ 1+(CantNodos i) + (CantNodos d)   Verdadero por int. ✓

Caso 2:

<=> 1+(if (CantNodos i) ≥ (CantNodos d) then (CantNodos i) else (CantNodos d)) ≤ 1+(CantNodos i) + (CantNodos d)

<=> 1+(if False then (CantNodos i) else (CantNodos d)) ≤ 1+(CantNodos i) + (CantNodos d)
{IfF}
<=> 1+ CantNodos(d) ≤ 1+(CantNodos i) + (CantNodos d)   Verdadero por int. ✓

Ejercicio 12 ★

Dados el tipo Polinomio definido en la práctica 1 y las siguientes funciones:

```
derivado :: Num a => Polinomio a -> Polinomio a
derivado poli = case poli of
   X         -> Cte 1
   Cte _     -> Cte 0
   Suma p q -> Suma (derivado p) (derivado q)
   Prod p q -> Suma (Prod (derivado p) q) (Prod (derivado q) p)


sinConstantesNegativas :: Num a => Polinomio a -> Polinomio a
sinConstantesNegativas = foldPoli True (>=0) (&&) (&&)

esRaiz :: Num a => a -> Polinomio a -> Bool
esRaiz n p = evaluar n p == 0
```

Demostrar las siguientes propiedades:

I. Num a => ∀ p::Polinomio a . ∀ q::Polinomio a . ∀ r::a . (esRaiz r p ⇒ esRaiz r (Prod p q))

II. Num a => ∀ p::Polinomio a . ∀ k::a . ∀ e::a .
evaluar e (derivado (Prod (Cte k) p)) = evaluar e (Prod (Cte k) (derivado p))

III. Num a => ∀ p::Polinomio a. (sinConstantesNegativas p⇒sinConstantesNegativas (derivado p))

La recursión utilizada en la definición de la función `derivado` ¿es estructural, primitiva o ninguna de las dos? → PRIMITIVA XQ USA EN SUBEST SIN RECURSION

→ NO ES ESTRUCTURAL.

→ NO ES GLOBAL XQ NO USA EN CADA PASO TODOS LOS RESULTADOS DE LAS RECURSIONES ANTERIORES.