

Práctica 2: Razonamiento Ecuacional e Inducción Estructural

Tomás Felipe Melli

May 12, 2025

Índice

1	Extensionalidad y Lemas de Generación	2
1.1	Ejercicio 1	2
1.2	Ejercicio 2	3
2	Inducción sobre Listas	4
2.1	Ejercicio 3	4
2.2	Ejercicio 4	7
2.3	Ejercicio 5	8
2.4	Ejercicio 6	8
3	Otras estructuras de datos	10
3.1	Ejercicio 9	10
3.2	Ejercicio 10	10
3.3	Ejercicio 11	12
3.4	Ejercicio 12	12

1 Extensionalidad y Lemas de Generación

1.1 Ejercicio 1

1	<code>intercambiar (x,y) = (y,x)</code>	<code>-{I}</code>
2	<code>espejar (Left x) = Right x</code>	<code>-{E1}</code>
3	<code>espejar (Right x) = Left x</code>	<code>-{E2}</code>
4	<code>asociarI (x,(y,z)) = ((x,y),z)</code>	<code>-{AI}</code>
5	<code>asociarD ((x,y),z)) = (x,(y,z))</code>	<code>-{AD}</code>
6	<code>flip f x y = f y x</code>	<code>-{F}</code>
7	<code>curry f x y = f (x,y)</code>	<code>-{C}</code>
8	<code>uncurry f (x,y) = f x y</code>	<code>-{U}</code>

Nos piden demostrar :

1. $\forall p :: (a,b).intercambiar (intercambiar p) = p$. Esta expresión está trabada, por tanto, recurrimos al lema de generación de pares que dice que *si* $\forall x :: a.\forall y :: b.P((x,y))$ *entonces* $\forall p :: (a,b).P(p)$. Con esto en mente, destrabamos y ..

$$intercambiar (intercambiar p) = p$$

$$\begin{aligned} & \text{Lema generación pares} \\ & \equiv intercambiar (intercambiar (x,y)) = (x,y) \\ & \stackrel{I}{\equiv} intercambiar (y,x) = (x,y) \\ & \stackrel{I}{\equiv} (x,y) = (x,y) \end{aligned}$$

2. $\forall p :: (a,(b,c)).asociarD (asociarI p) = p$. Como antes, está trabada. Por el principio de inducción sobre pares podemos expresar a que la propiedad vale sobre la tupla y por tanto...

$$asociarD (asociarI p) = p$$

$$\begin{aligned} & \text{Lema generación pares} \\ & \equiv asociarD (asociarI (x,y)) = (x,y) \end{aligned}$$

Sigue trabado el y , por tanto, aplicamos de nuevo el lema.

$$\begin{aligned} & \text{Lema generación pares} \\ & \equiv asociarD (asociarI (x,(i,j))) = (x,(i,j)) \\ & \stackrel{AI}{\equiv} asociarD ((x,i),j) = (x,(i,j)) \\ & \stackrel{AD}{\equiv} (x,(i,j)) = (x,(i,j)) \end{aligned}$$

3. $\forall p :: \text{Either } a \ b. espejar (espejar p) = p$. Esta expresión está trabada. Podemos utilizar el *lema de generación para Sumassi* $p :: \text{Either } a \ b$ *entonces* :

- o bien $\exists x :: a.p = Left x$
- o bien $\exists y :: b.p = Right y$

Con esto en mente, separamos en casos :

•

$$\begin{aligned} & \text{Lema} \\ & \equiv espejar (espejar (Left x)) = (Left x) \\ & \stackrel{E1}{\equiv} espejar (Right x) = (Left x) \\ & \stackrel{E2}{\equiv} (Left x) = (Left x) \end{aligned}$$

•

$$\begin{aligned} & \text{Lema} \\ & \equiv espejar (espejar (Right y)) = (Right y) \\ & \stackrel{E2}{\equiv} espejar (Left y) = (Right y) \\ & \stackrel{E1}{\equiv} (Right y) = (Right y) \end{aligned}$$

4. $\forall f :: a- > b- > c. \forall x :: a. \forall y :: b. \text{flip} (\text{flip } f) x y = f x y$. En este caso, queremos demostrar algo sobre el comportamiento y no sobre la representación interna, por tal motivo tenemos que pensar en **Extensionalidad**.

$$\text{flip} (\text{flip } f) x y = f x y$$

$$\stackrel{F}{\equiv} \text{flip } f y x = f x y$$

$$\stackrel{F}{\equiv} f x y = f x y$$

5. $\forall f :: a- > b- > c. \forall x :: a. \forall y :: b. \text{curry} (\text{uncurry } f) x y = f x y$. Tenemos que probar cosas sobre el comportamiento de ambas y por ello, utilizamos el principio de extensionalidad para probar esta igualdad.

$$\text{curry} (\text{uncurry } f) x y = f x y$$

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

$$\stackrel{U}{\equiv} f x y = f x y$$

1.2 Ejercicio 2

1. $\text{flip} . \text{flip} = \text{id}$. Para demostrar esta igualdad, seguimos el principio de Extensionalidad funcional que nos dice que este problema se reduce a resolver algo del tipo $f = g$. Para ello, nos proponemos $\forall x :: (a- > b- > c) - > b- > a- > c$ y

$$\text{flip} . \text{flip } x = \text{id } x$$

$$\stackrel{(\cdot)}{\equiv} (\text{flip} . \text{flip}) x = \text{id } x$$

$$\stackrel{(\cdot)}{\equiv} \text{flip} (\text{flip } x) = \text{id } x$$

$$\stackrel{\text{por ejercicio 1.5}}{\equiv} x = \text{id } x$$

$$\stackrel{\text{id}}{\equiv} \text{id } x = \text{id } x$$

2. $\forall f :: (a, b) - > c. \text{uncurry} (\text{curry } f) = f$ Definimos $\forall x :: a. \forall y :: b$

$$\equiv \text{uncurry} (\text{curry } f) (x, y) = f (x, y)$$

$$\stackrel{\text{uncurry}}{\equiv} \text{curry } f x y = f (x, y)$$

$$\stackrel{\text{curry}}{\equiv} f (x, y) = f (x, y)$$

3. $\text{flip } \text{const} = \text{const } \text{id}$ Podemos definir $\forall x :: a. \forall y :: b$ y tomar entonces:

$$\text{flip } \text{const } x y = \text{const } \text{id } x y$$

$$\stackrel{\text{flip}}{\equiv} \text{const } y x = \text{const } \text{id } x y$$

$$\stackrel{\text{const}}{\equiv} y = \text{const } \text{id } x y$$

$$\stackrel{\text{const}}{\equiv} y = \text{id } x \quad \text{---chequear}$$

4. $\forall f :: a- > b. \forall g :: b- > c. \forall h :: c- > d$. queremos ver que :

$$((h . g) . f) = (h . (g . f))$$

Por principio de extensionalidad funcional bastaría con ver que $\forall x :: a :$

$$((h . g) . f) x = (h . (g . f)) x$$

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

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

2 Inducción sobre Listas

2.1 Ejercicio 3

Se tienen las siguientes funciones:

```
1 length :: [a] -> Int
2 {L0} length [] = 0
3 {L1} length (x:xs) = 1 + length xs
4
5 duplicar :: [a] -> [a]
6 {D0} duplicar [] = []
7 {D1} duplicar (x:xs) = x : x : duplicar xs
8
9 (++) :: [a] -> [a] -> [a]
10 {++0} [] ++ ys = ys
11 {++1} (x:xs) ++ ys = x : (xs ++ ys)
12
13 append :: [a] -> [a] -> [a]
14 {A0} append xs ys = foldr (:) ys xs
15
16 ponerAlFinal :: a -> [a] -> [a]
17 {P0} ponerAlFinal x = foldr (:) (x:[])
18
19 reverse :: [a] -> [a]
20 {R0} reverse = foldl (flip (:)) []
```

1. $\forall xs :: [a].$

$$length (duplicar xs) = 2 \times length xs$$

Queremos demostrar cierta propiedad sobre listas, por ello, miremos la definición del tipo :

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

Tenemos que probar que la propiedad P vale para $P[]$ y que también vale $\forall x :: a. \forall xs :: [a]. (P(xs) \implies P(x : xs))$. Por inducción estructural sobre listas decimos que

- Caso base : $P[]$ vale :

$$length (duplicar []) = 2 \times length []$$

$$\stackrel{D0}{\equiv} \stackrel{L0}{-} length [] = 2 \times 0$$

$$\stackrel{L0}{\equiv} 0 = 0$$

- Caso inductivo : tomamos como **H.I**

$$P(xs) \equiv (length (duplicar xs) = 2 \times length xs)$$

Entonces, como tomamos la hipótesis como verdadera, pasamos a probar que vale la propiedad para

$$P(x : xs) \equiv length (duplicar (x : xs)) = 2 \times length (x : xs)$$

$$length (duplicar (x : xs))$$

$$\stackrel{D1}{\equiv} length (x : x : duplicar xs)$$

$$\stackrel{L1}{\equiv} 1 + length (x : duplicar xs)$$

$$\stackrel{L1}{\equiv} 1 + 1 + length (duplicar xs)$$

$$\stackrel{HI}{\equiv} 1 + 1 + (2 \times length xs)$$

$$\stackrel{(*)}{\equiv} 1 + 1 + (length xs + length xs)$$

$$\stackrel{(+)}{\equiv} (1 + length xs) + (1 + length xs)$$

$$\stackrel{L1}{\equiv} (length (x : xs)) + (length (x : xs))$$

$$\equiv 2 \times length (x : xs)$$

2. $\forall xs :: [a]. \forall ys :: [a].$ Queremos probar que

$$length (xs ++ ys) = length xs + length ys$$

Para ello, tenemos que utilizar inducción estructural sobre el tipo :

- Vale la propiedad para $P([])$

$$length ([] ++ []) = length [] + length []$$

$$\stackrel{++0}{\equiv} length ([])$$

$$\stackrel{L0}{\equiv} 0$$

$$\stackrel{L0}{\equiv} length[]$$

$$\stackrel{L0}{\equiv} length[] + length[]$$

- Queremos ver que $\forall x :: a. P(xs) \implies P(x : xs)$ asumiendo como **HI**

$$P(xs) \equiv length (xs ++ ys) = length xs + length ys$$

Queremos probar que :

$$length ((x : xs) ++ ys) = length (x : xs) + length ys$$

$$\stackrel{++1}{\equiv} length (x : (xs ++ ys))$$

$$\stackrel{L1}{\equiv} 1 + length (xs ++ ys)$$

$$\stackrel{HI}{\equiv} 1 + length xs + length ys$$

$$\stackrel{L1}{\equiv} length (x : xs) + length ys$$

3. $\forall xs :: [a]. \forall x :: a.$ Queremos ver que valga

$$append [x] xs = x : xs$$

$$\stackrel{A0}{\equiv} foldr (:) xs [x]$$

$$\stackrel{re-escritura}{\equiv} foldr (:) xs (x : [])$$

$$\stackrel{foldr \text{ caso recursivo}}{\equiv} (:) x (foldr (:) xs [])$$

$$\stackrel{foldr \text{ caso base}}{\equiv} (:) x xs$$

$$\equiv x (:) xs$$

Sale directo (??)

4. $\forall xs :: [a]. \forall f :: (a \rightarrow b).$ Queremos probar que

$$length (map f xs) = length xs$$

Podemos pensar en demostrarlo con inducción sobre el tipo, entonces postulamos :

(a) Caso Base : la propiedad vale para ... $P([])$

$$length (map f []) = length []$$

$$\stackrel{map-caso \text{ base}}{\equiv} length [] = length []$$

(b) Caso Recursivo, queremos ver que si la propiedad vale para cierta lista, también vale para esa lista más un elemento.

$$P(xs) \implies P(x : xs)$$

$$length (map f (x : xs)) = length (x : xs)$$

$$\stackrel{map}{\equiv} length (f x : map f xs)$$

$$\stackrel{L1}{\equiv} 1 + length (map f xs)$$

$$\stackrel{H.I}{\equiv} 1 + length xs$$

$$\stackrel{L1}{\equiv} length (x : xs)$$

5. $\forall xs :: [a]. \forall p :: a \rightarrow Bool. \forall e :: a. \text{Queremos ver que vale la siguiente igualdad (asumiendo que } Eq =_i \text{ a)}$

$$((elem\ e\ (filter\ p\ xs)) \implies (elem\ e\ xs))$$

Para demostrarlo, lo vamos a hacer sobre inducción estructural sobre el tipo.

(a) Caso Base : $P([])$

$$\begin{aligned} & ((elem\ e\ (filter\ p\ [])) \implies (elem\ e\ [])) \\ & \stackrel{filter-caso\ base}{\equiv} ((elem\ e\ []) \implies (elem\ e\ [])) \end{aligned}$$

Falso \implies Verdadero

(b) Caso Inductivo: $P(xs) \implies P(x : xs)$

$$\begin{aligned} & ((elem\ e\ (filter\ p\ (x : xs))) \implies (elem\ e\ (x : xs))) \\ & \stackrel{filter}{\equiv} (elem\ e\ (if\ p\ x\ then\ x : (filter\ p\ xs)\ else\ (filter\ p\ xs))) \end{aligned}$$

El tema es que tenemos **dos casos**, si el elemento x cumple el predicado p o si no lo cumple. Para destrabar esto, usamos el **principio de inducción sobre Bool**

i. Tomamos que $p\ x\ es\ True$

$$\stackrel{filter}{\equiv} (elem\ e\ (x : (filter\ p\ xs)))$$

Pero volvemos a tener problemas, je :). Porque puede ser que x sea e o no, y e se encuentre en xs . Otra vez, separamos los casos

- Si $e == x$ esto implica que efectivamente $elem\ e\ (x : xs)$.
- Si $e \in xs$ vale por H.I

ii. Tomamos que $p\ x\ es\ False$

$$\stackrel{filter}{\equiv} (elem\ e\ (filter\ p\ xs))$$

Que por H.I esto vale.

6. $\forall xs :: [a]. \forall x :: a. \text{Queremos probar que}$

$$ponerAlFinal\ x\ xs = xs ++ (x : [])$$

La idea es análoga a lo que venimos haciendo. Inducción sobre el tipo.

(a) $P([])$

$$\begin{aligned} & ponerAlFinal\ x\ [] = [] ++ (x : []) \\ & \stackrel{P0}{\equiv} foldr\ (\cdot)\ (x : [])\ [] \\ & \stackrel{foldr\ caso\ base}{\equiv} (x : []) \\ & \equiv [x] \\ & \stackrel{++0}{\equiv} [] ++ (x : []) \end{aligned}$$

(b) $P(xs) \implies P(x : xs)$

$$\begin{aligned} & ponerAlFinal\ y\ (x : xs) = (x : xs) ++ (y : []) \\ & \equiv ponerAlFinal\ y\ (x : xs) \\ & \stackrel{P0}{\equiv} foldr\ (\cdot)\ (y : [])\ (x : xs) \\ & \stackrel{foldr}{\equiv} (\cdot)\ x\ (foldr\ (\cdot)\ (y : [])\ xs) \\ & \stackrel{P0}{\equiv} (\cdot)\ x\ (ponerAlFinal\ y\ xs) \\ & \stackrel{HI}{\equiv} (\cdot)\ x\ (xs ++ (y : [])) \\ & \stackrel{++1}{\equiv} (x : xs) ++ (y : []) \end{aligned}$$

7.

$$reverse = foldr (x \text{ rec } -> \text{rec} ++ (x : [])) []$$

Por principio de extensionalidad funcional queremos ver que $\forall xs :: [a]$

$$reverse\ xs = foldr (x \text{ rec } -> \text{rec} ++ (x : [])) []\ xs$$

Primero queremos ver que valga esta propiedad para el caso base, ya que hacemos inducción estructural sobre el tipo.

(a) $P([])$

$$\begin{aligned} reverse\ [] &= foldr (x \text{ rec } -> \text{rec} ++ (x : [])) []\ [] \\ &\stackrel{foldr\ caso\ base}{=} reverse\ [] = [] \\ &\stackrel{R0}{=} foldl (flip\ (:))\ []\ [] = [] \\ &\stackrel{foldl\ caso\ base}{=} [] = [] \end{aligned}$$

(b) $P(xs) \implies P(x : xs)$

$$\begin{aligned} reverse\ (x : xs) &= foldr (y \text{ rec } -> \text{rec} ++ (y : [])) (x : xs) \\ &\stackrel{R0}{=} foldl (flip\ (:))\ []\ (x : xs) \\ &\stackrel{foldl}{=} foldl (flip\ (:))\ ((flip\ (:))\ []\ x)\ xs \end{aligned}$$

(c) $P(xs) \implies P(x : xs)$ Del otro lado porque me trabé.

$$\begin{aligned} reverse\ (x : xs) &= foldr (y \text{ rec } -> \text{rec} ++ (y : [])) []\ (x : xs) \\ &\stackrel{foldr}{=} (y \text{ rec } -> \text{rec} ++ (y : []))\ x\ (foldr (y \text{ rec } -> \text{rec} ++ (y : [])) []\ xs) \\ &\stackrel{HI}{=} (y \text{ rec } -> \text{rec} ++ (y : []))\ x\ (reverse\ xs) \\ &\stackrel{\beta}{=} (\text{rec } -> \text{rec} ++ (x : []))\ (reverse\ xs) \\ &\stackrel{\beta}{=} reverse\ xs ++ (x : []) \end{aligned}$$

8. $\forall xs :: [a]. \forall x :: a$

$$head\ (reverse\ (ponerAlFinal\ x\ xs)) = x$$

2.2 Ejercicio 4

1. Queremos demostrar la siguiente igualdad $reverse \ . \ reverse = id$ Por principio de de extensionalidad funcional, bastaría con probar que $\forall xs :: [a] \quad reverse \ . \ reverse\ xs = id\ xs$ Que podremos demostrar mediante el uso de inducción estructural sobre listas.

$$reverse \ . \ reverse\ xs = id\ xs$$

• Caso Base : $P([])$

$$\begin{aligned} reverse \ . \ reverse\ [] &= id\ [] \\ &\stackrel{(\cdot)}{=} reverse\ (reverse\ []) \\ &\stackrel{R0}{=} reverse\ (foldl\ (flip\ (:))\ []\ []) \\ &\stackrel{foldl}{=} reverse\ [] \\ &\stackrel{R0}{=} (foldl\ (flip\ (:))\ []\ []) \\ &\stackrel{foldl}{=} [] \\ &\stackrel{id}{=} id\ [] \end{aligned}$$

• $P(xs) \implies P(x : xs)$

$$\begin{aligned} reverse \ . \ reverse\ (x : xs) &= id\ (x : xs) \\ &\stackrel{(\cdot)}{=} reverse\ (reverse\ (x : xs)) \\ &\stackrel{R0}{=} reverse\ (foldl\ (flip\ (:))\ []\ (x : xs)) \\ &\stackrel{foldl}{=} reverse\ (foldl\ (flip\ (:))\ ((flip\ (:))\ []\ x)\ xs) \end{aligned}$$

2.3 Ejercicio 5

```

1 zip :: [a] -> [b] -> [(a,b)]
2 {Z0} zip = foldr (\x rec ys -> if null ys then [] else (x, head ys) : rec (tail ys)) (const [])
3
4 zip' :: [a] -> [b] -> [(a,b)]
5 {Z0'} zip' [] ys = []
6 {Z1'} zip' (x:xs) ys = if null ys then [] else (x, head ys):zip' xs (tail ys)

```

Queremos probar que $zip = zip'$

Para ello, por principio de extensionalidad bastaría probar que $\forall xs, ys :: [a]. \quad zip\ xs\ ys = zip'\ xs\ ys$. Para probar esta igualdad hacemos uso de inducción estructural sobre listas. Y por ello planteamos

1. Caso Base : $P([])$

$$\begin{aligned}
 & zip\ []\ ys = zip'\ []\ ys \\
 & \stackrel{Z0}{=} foldr\ (i\ rec\ is \rightarrow if\ null\ is\ then\ []\ else\ (i,\ head\ is) : rec\ (tail\ is))\ (const\ [])\ []\ ys \\
 & \stackrel{foldr}{=} const\ []\ ys \\
 & \stackrel{const}{=} [] \\
 & \stackrel{Z0'}{=} zip'\ []\ ys
 \end{aligned}$$

2. $\forall x :: a \quad P(xs) \implies P(x:xs)$. Queremos probar que vale

$$\begin{aligned}
 & zip\ (x:xs)\ ys = zip'\ (x:xs)\ ys \\
 & \equiv zip\ (x:xs)\ ys \\
 & \stackrel{Z0}{=} foldr\ (i\ rec\ is \rightarrow if\ null\ is\ then\ []\ else\ (i,\ head\ is) : rec\ (tail\ is))\ (const\ [])\ (x:xs)\ ys \\
 & \stackrel{foldr}{=} (i\ rec\ is \rightarrow if\ null\ is\ then\ []\ else\ (i,\ head\ is) : rec\ (tail\ is))\ x \\
 & \quad (foldr\ (i\ rec\ is \rightarrow if\ null\ is\ then\ []\ else\ (i,\ head\ is) : rec\ (tail\ is))\ (const\ [])\ xs)\ ys \\
 & \stackrel{Z0}{=}
 \end{aligned}$$

2.4 Ejercicio 6

```

1 nub :: Eq a => [a] -> [a]
2 {N0} nub [] = []
3 {N1} nub (x:xs) = x : filter (\y -> x /= y) (nub xs)
4
5 union :: Eq a => [a] -> [a] -> [a]
6 {U0} union xs ys = nub (xs++ys)
7
8 intersect :: Eq a => [a] -> [a] -> [a]
9 {I0} intersect xs ys = filter (\e -> elem e ys) xs

```

Tenemos que indicar si las siguientes propiedades son verdaderas o falsas. En caso de ser verdaderas, probarlas ; caso contrario, contraejemplo.

1. $Eq\ a \implies \forall xs :: [a]. \forall e :: a. \forall p :: a \rightarrow Bool. \quad elem\ e\ xs \ \&\&\ p\ e = elem\ e\ (filter\ p\ xs)$ Es verdadero, si el elemento está en la lista y cumple con el predicado, si construimos una lista con filter aplicando ese predicado, efectivamente va a estar en la lista resultante el elemento. La demo es muy similar al ejercicio 3.5.
2. $Eq\ a \implies \forall xs :: [a]. \forall e :: a. \quad elem\ e\ xs = elem\ e\ (nub\ xs)$
 Qué hace *nub* ? elimina las apariciones repetidas de cada elemento de la lista que recibe como parámetro. Por tanto, la igualdad se cumple, dado que el elemento está en la lista y por tanto, aparece al menos una vez. Como consecuencia, estará en la lista generada por *nub*. Queremos ver entonces que vale. Y por ello, por inducción estructural sobre listas...

$$P(xs) \equiv elem\ e\ xs = elem\ e\ (nub\ xs)$$

- Caso Base $P([])$

$$\begin{aligned}
elem\ e\ [] &= elem\ e\ (nub\ []) \\
&\equiv elem\ e\ [] \\
&\stackrel{elem}{\equiv} False \\
&\stackrel{elem}{\equiv} elem\ [] \\
&\stackrel{N0}{\equiv} elem\ (nub\ [])
\end{aligned}$$

- Caso Inductivo $P(xs) \implies P(x : xs) \quad \forall x :: a$

$$\begin{aligned}
elem\ e\ (x : xs) &= elem\ e\ (nub\ (x : xs)) \\
&\equiv elem\ e\ (x : xs) \\
&\stackrel{elem}{\equiv} e == x \parallel elem\ e\ xs
\end{aligned}$$

Esto nos lleva a **dos casos**. Que por principio de inducción sobre booleanos ...

- Si $e == x$
- * Si **True**

$$\begin{aligned}
&\equiv True \parallel elem\ e\ xs \\
&\equiv True \\
&\stackrel{elem}{\equiv} elem\ e\ (nub\ (x : xs))
\end{aligned}$$

- * Si **False**

$$\begin{aligned}
&\equiv False \parallel elem\ e\ xs \\
&\equiv elem\ e\ xs
\end{aligned}$$

y por tanto dependerá de...

$$\begin{aligned}
&\equiv elem\ e\ xs \\
&\stackrel{HI}{\equiv} elem\ e\ (nub\ xs)
\end{aligned}$$

3. $Eq\ a \implies \forall xs :: [a]. \forall ys :: [a]. \forall e :: a. \quad elem\ e\ (union\ xs\ ys) = (elem\ e\ xs) \parallel (elem\ e\ ys)$. En castellano es, el elemento e pertenece a la unión sin repetidos entonces o pertenece a xs o pertenece a ys (no es un \vee exclusivo ya que puede estar en las dos pero no estará en la filtrada). Lo cual es cierto.
4. $Eq\ a \implies \forall xs :: [a]. \forall ys :: [a]. \forall e :: a. \quad elem\ e\ (intersect\ xs\ ys) = (elem\ e\ xs) \&\& (elem\ e\ ys)$. Nos dice que si un elemento $e \in xs \wedge e \in ys$ en la lista resultante estará. Queremos demostrar que debe estar en ambas.
5. $Eq\ a \implies \forall xs :: [a]. \forall ys :: [a]. \quad length\ (union\ xs\ ys) = length\ xs + length\ ys$. Esta expresión dice que la longitud de la **union(U0)** de dos listas es la misma que la suma de las longitudes de cada lista. Con esta definición de Unión, esto es falso si hay elementos en común.

$$Sea\ xs = [1, 2, 3]\ y\ ys = [1, 2, 3]$$

$$\begin{aligned}
length\ (union\ xs\ ys) &= length\ xs + length\ ys \\
&\stackrel{U0}{\equiv} length\ (nub\ (xs ++ ys)) = length\ xs + length\ ys \\
&\stackrel{N1}{\equiv} length\ (nub\ (xs ++ ys)) = length\ xs + length\ ys
\end{aligned}$$

Que en este ejemplo sabemos que $[1, 2, 3] ++ [1, 2, 3] = [1, 2, 3, 1, 2, 3]$ y que si hacemos $nub\ [1, 2, 3, 1, 2, 3] = [1, 2, 3]...$

$$\equiv length\ [1, 2, 3] = length\ [1, 2, 3] + length\ [1, 2, 3]$$

$$\equiv 3 = 6$$

Lo cual es **absurdo**.

6. $Eq\ a \implies \forall xs :: [a]. \forall ys :: [a]. \quad length\ (union\ xs\ ys) \leq length\ xs + length\ ys$. Lo que propone la expresión es que la longitud de la lista resultante de la unión sin repetidos es menor o igual a la suma de las listas que se unirán. Existen dos casos que tal vez tengan relevancia analizar : el anterior, listas iguales, sabemos que vale, que es mayor la suma de las longitudes. Y el caso en que son diferentes, como $xs = [1]$ y $ys = [2]$ con lo cual no da que es exactamente igual. Esta expresión tiene sentido y es Verdadera.

3 Otras estructuras de datos

3.1 Ejercicio 9

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

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

Primero, nos traemos las funciones :

```
1 data AB a = Nil | Bin (AB a) r (AB a)
2
3 altura :: AB a -> Int
4 {A} altura = foldAB 0 (\recI r recD -> 1 + max recI recD)
5
6 cantNodos :: AB a -> Int
7 {C} cantNodos = foldAB 0 (\recI r recD -> 1 + recI + recD)
```

Queremos ver entonces que $\forall x :: AB\ a. \quad altura\ x \leq cantNodos\ x$. Vamos a proceder por inducción sobre **AB a** :

$$\forall x :: AB\ a. \quad P(x) = altura\ x \leq cantNodos\ x$$

- Caso Base :

$$P(Nil) = altura\ Nil \leq cantNodos\ Nil$$

$$\begin{matrix} \{A\}\{C\} \\ \equiv \\ 0 \leq 0 \end{matrix}$$

- Paso Inductivo : queremos ver que $\forall i : AB\ a, r : a, d : AB\ a$ (lo que hacemos es declarar los tipos de las variables que vamos necesitar en la demo) se cumple que $P(i) \wedge P(d) \implies P(Bin\ i\ r\ d)$. Con esto **asumimos** $P(i) \wedge P(d)$ (decimos explícitamente que el antecedente vale) para luego demostrar que la propiedad :

$P(Bin\ i\ r\ d)$ vale. Decimos entonces que queremos ver que $(QVQ) \forall x : AB\ a$

$$\begin{aligned} P(Bin\ i\ r\ d) &= altura\ (Bin\ i\ r\ d) \leq cantNodos\ (Bin\ i\ r\ d) \\ &\stackrel{\{A\}}{\equiv} foldAB\ 0\ (\backslash recI\ r\ recD \rightarrow 1 + \max recI\ recD)\ (Bin\ i\ r\ d) \leq cantNodos\ (Bin\ i\ r\ d) \\ &\stackrel{\beta}{\equiv} 1 + \max\ altura(i)\ altura(d) \leq cantNodos\ (Bin\ i\ r\ d) \\ &\stackrel{\{C\}}{\equiv} 1 + \max\ altura(i)\ altura(d) \leq foldAB\ 0\ (\backslash recI\ r\ recD \rightarrow 1 + recI + recD)\ (Bin\ i\ r\ d) \\ &\stackrel{\beta}{\equiv} 1 + \max\ altura(i)\ altura(d) \leq 1 + cantNodos(i) + cantNodos(d) \\ &\stackrel{\{HI^*\}}{\equiv} \max\ (cantNodos(i), cantNodos(d)) \leq cantNodos(i) + cantNodos(d) \\ &\stackrel{N}{\equiv} \max\ (cantNodos(i), cantNodos(d)) \leq cantNodos(i) + cantNodos(d) \end{aligned}$$

Aclaración * : cuando aplicamos HI, vemos que el subárbol izquierdo i cumple la propiedad así como el subárbol d. Esto es equivalente a $P(i) = altura\ i \leq cantNodos\ i$ y $P(d) = altura\ d \leq cantNodos\ d$, hacemos los reemplazos adecuadamente. Luego de esto, es ver cómo se comportan los enteros. Si el mximo entre dos números es menor o igual a la suma de estos. Lo cual, es válido para los naturales.

3.2 Ejercicio 10

Dada la siguiente función:

```
1 truncar :: AB a -> Int -> AB a
2 {T0} truncar Nil _ = Nil
3 {T1} truncar (Bin i r d) n = if n == 0 then Nil else Bin (truncar i (n-1)) r (truncar d (n-1))
```

Y los siguientes lemas:

1. $\forall x :: Int. \forall y :: Int. \forall z :: Int. \max(\min(x, y), \min(x, z)) = \min(x, \max(y, z))$
2. $\forall x :: Int. \forall y :: Int. \forall z :: Int. z + \min(x, y) = \min(z + x, z + y)$

Demostrar las siguientes propiedades:

- i. $\forall t :: AB\ a. altura(t) \geq 0$

ii. $\forall t :: AB\ a. \forall n :: Int. (n \geq 0 \Rightarrow (\text{altura}(\text{truncar } t\ n) = \min n\ (\text{altura } t)))$

Vamos a probar ...

1. Lo que queremos probar es $\forall t :: AB\ a. P(t) = \text{altura}(t) \geq 0$. Por tanto, por inducción sobre **AB a**

- Caso Base :

$$\begin{aligned} P(\text{Nil}) &= \text{altura}(\text{Nil}) \geq 0 \\ &\stackrel{\{A\}}{=} \text{foldAB } 0\ (\backslash \text{recI } r\ \text{recD} \rightarrow 1 + \max \text{recI } \text{recD})\ \text{Nil} \geq 0 \\ &\stackrel{\beta}{=} 0 \geq 0 \end{aligned}$$

- Paso Inductivo : Antes de comenzar, tenemos que definir qué es lo que queremos probar, y qué es lo que vale (las HI) para poder lograrlo. Los pasos como venimos viendo son : definir los tipos de las variables involucradas en la proposición , escribir la proposición a probar, explícitamente decir que vale el antecedente y luego proceder a demostrar.

- (a) $\forall i : AB\ a, r : a, d : AB\ a$
- (b) $P(i) \wedge P(d) \implies P(\text{Bin } i\ r\ d)$
- (c) Asumimos que valen $P(i) \wedge P(d)$
- (d) Queremos entonces probar que : $P(\text{Bin } i\ r\ d) = \text{altura}(\text{Bin } i\ r\ d) \geq 0$

$$\begin{aligned} P(\text{Bin } i\ r\ d) &= \text{altura}(\text{Bin } i\ r\ d) \geq 0 \\ &\stackrel{\{A\}}{=} \text{foldAB } 0\ (\backslash \text{recI } r\ \text{recD} \rightarrow 1 + \max \text{recI } \text{recD})(\text{Bin } i\ r\ d) \geq 0 \\ &\stackrel{\beta}{=} 1 + \max(\text{altura}(i), \text{altura}(d)) \geq 0 \\ &\stackrel{HI^*}{=} 1 \geq 0 \end{aligned}$$

Aclaración HI * : sabemos por HI que $\text{altura}(i) \geq 0$ y $\text{altura}(d) \geq 0$.

2. Lo que queremos probar es $\forall t :: AB\ a. \forall n :: Int. P(t) = n \geq 0 \Rightarrow (\text{altura}(\text{truncar } t\ n) = \min n\ (\text{altura } t))$. Por tanto, con inducción sobre **AB a**

- Caso Base :

$$\begin{aligned} P(\text{Nil}) &= n \geq 0 \Rightarrow (\text{altura}(\text{truncar } \text{Nil } n) = \min n\ (\text{altura } \text{Nil})) \\ &\stackrel{\{T0\}}{=} n \geq 0 \Rightarrow (\text{altura}(\text{Nil}) = \min n\ (\text{altura } \text{Nil})) \\ &\stackrel{\{A\}}{=} n \geq 0 \Rightarrow (0 = \min n\ 0) \\ &\stackrel{\{min\}}{=} n \geq 0 \Rightarrow (0 = 0) \end{aligned}$$

- Paso inductivo : $\forall i : AB\ a, r : a, d : AB\ a$ queremos probar que $P(i) \wedge P(d) \implies P(\text{Bin } i\ r\ d)$. Asumimos que vale el antecedente y entonces enunciamos

$$P(\text{Bin } i\ r\ d) = n \geq 0 \Rightarrow (\text{altura}(\text{truncar } (\text{Bin } i\ r\ d)\ n) = \min n\ (\text{altura } (\text{Bin } i\ r\ d)))$$

Separamos en casos :

- Caso $N == 0$

$$\begin{aligned} P(\text{Bin } i\ r\ d) &= n = 0 \Rightarrow (\text{altura}(\text{truncar } (\text{Bin } i\ r\ d)\ 0) = \min 0\ (\text{altura } (\text{Bin } i\ r\ d))) \\ &\stackrel{\{T1\}}{=} n = 0 \Rightarrow (\text{altura}(\text{Nil}) = \min 0\ (\text{altura } (\text{Bin } i\ r\ d))) \\ &\stackrel{\{Min\}}{=} n = 0 \Rightarrow (\text{altura}(\text{Nil}) = 0) \\ &\stackrel{\{A\}}{=} n = 0 \Rightarrow (0 = 0) \end{aligned}$$

- Caso $N > 0$

$$\begin{aligned} P(\text{Bin } i\ r\ d) &= n > 0 \Rightarrow (\text{altura}(\text{truncar } (\text{Bin } i\ r\ d)\ n) = \min n\ (\text{altura } (\text{Bin } i\ r\ d))) \\ &\stackrel{\{T1\}}{=} n > 0 \Rightarrow (\text{altura}(\text{Bin } (\text{truncar } i\ (n-1))\ r\ (\text{truncar } i\ (n-1)))) = \min n\ (\text{altura } (\text{Bin } i\ r\ d))) \\ &\stackrel{\{A\}}{=} n > 0 \Rightarrow (\text{foldAB } 0\ (\backslash \text{recI } r\ \text{recD} \rightarrow 1 + \max \text{recI } \text{recD})\ (\text{Bin } (\text{truncar } i\ (n-1))\ r\ (\text{truncar } i\ (n-1)))) = \min n\ (1 + \max(\text{altura}(i), \text{altura}(d))) \\ &\stackrel{\{T1\}}{=} n > 0 \Rightarrow (1 + \max(\text{altura}(\text{truncar } i\ (n-1)), \text{altura}(\text{truncar } d\ (n-1)))) = \min n\ (1 + \max(\text{altura}(i), \text{altura}(d))) \end{aligned}$$

3.3 Ejercicio 11

Considerar las siguientes funciones:

```

1 inorder :: AB a -> [a]
2 {I0} inorder = foldAB [] (\ri x rd -> ri ++ (x:rd))
3
4 elemAB :: Eq a => a -> AB a -> Bool
5 {A0} elemAB e = foldAB False (\ri x rd -> (e == x) || ri || rd)
6
7 elem :: Eq a => [a] -> Bool
8 {E0} elem e = foldr (\x rec -> (e == x) || rec) False

```

Demostrar la siguiente propiedad:

$$\text{Eq } a \Rightarrow \forall e :: a, \text{elemAB } e = \text{elem } e . \text{inorder}$$

Si miramos, no pusieron explícitamente el árbol

$$\text{Eq } a \Rightarrow \forall e :: a, \text{elemAB } e \text{ ab} = \text{elem } e . \text{inorder ab}$$

Con esto, vamos a querer probar esta igualdad por inducción sobre **AB a**.

Para ello, decimos que $\forall t : AB \ a.P(t) = \text{elemAB } e \ t = \text{elem } e . \text{inorder } t$.

- Caso Base :

$$\begin{aligned}
 P(\text{Nil}) &= \text{elemAB } e \ \text{Nil} = \text{elem } e . \text{inorder } \text{Nil} \\
 &\stackrel{\{A0\}}{=} \text{foldAB } \text{False} \ (\backslash ri \ x \ rd \rightarrow (e == x) \ || \ ri \ || \ rd) \ \text{Nil} = \text{elem } e . \text{inorder } \text{Nil} \\
 &\stackrel{\beta}{=} \text{False} = \text{elem } e . \text{inorder } \text{Nil} \\
 &\stackrel{\{I0\}}{=} \text{False} = \text{elem } e . \text{foldAB } [] (\backslash ri \ x \ rd \rightarrow ri ++ (x:rd)) \ \text{Nil} \\
 &\stackrel{\beta}{=} \text{False} = \text{elem } e . [] \\
 &\stackrel{\{E0\}}{=} \text{False} = \text{foldr } (\backslash x \ rec \rightarrow (e == x) || rec) \ \text{False } e . [] \\
 &\stackrel{(\cdot)}{=} \text{False} = (\text{foldr } (\backslash x \ rec \rightarrow (e == x) || rec) \ \text{False } e) [] \\
 &\stackrel{\beta}{=} \text{False} = \text{False}
 \end{aligned}$$

- Paso Inductivo : vamos a querer probar que $\forall i : AB \ a, r : a, d : AB \ a$ asumiendo como premisas que $P(i) \wedge P(d)$ valen, entonces $P(\text{Bin } i \ r \ d)$. Queremos probar entonces

$$\begin{aligned}
 P(\text{Bin } i \ r \ d) &= \text{elemAB } e \ (\text{Bin } i \ r \ d) = \text{elem } e . \text{inorder } (\text{Bin } i \ r \ d) \\
 &\stackrel{\{A0\}}{=} \text{foldAB } \text{False} \ (\backslash ri \ r \ rd \rightarrow (e == x) || ri || rd) \ (\text{Bin } i \ r \ d) \\
 &\stackrel{\beta}{=} (e == x) || \text{elemAB}(i) || \text{elemAB}(d) \ (\text{Bin } i \ r \ d)
 \end{aligned}$$

3.4 Ejercicio 12

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

```

1 derivado :: Num a => Polinomio a -> Polinomio a
2 derivado poli = case poli of
3   X          -> Cte 1
4   Cte _      -> Cte 0
5   Suma p q   -> Suma (derivado p) (derivado q)
6   Prod p q   -> Suma (Prod (derivado p) q) (Prod (derivado q) p)
7
8 sinConstantesNegativas :: Num a => Polinomio a -> Polinomio a
9 sinConstantesNegativas = foldPoli True (>=0) (&&) (&&)
10
11 esRaiz :: Num a => a -> Polinomio a -> Bool
12 esRaiz n p = evaluar n p == 0

```

Nos piden demostrar :

1. $\text{Num } a \Rightarrow \forall p :: \text{Polinomio } a. \forall q :: \text{Polinomio } a. \forall r :: a. (\text{esRaiz } r \ p \implies \text{esRaiz } r \ (\text{Prod } p \ q))$
2. $\text{Num } a \Rightarrow \forall p :: \text{Polinomio } a. \forall k :: a. \forall e :: a. \text{evaluar } e \ (\text{derivado } (\text{Prod } (\text{Cte } k) \ p)) = \text{evaluar } e \ (\text{Prod } (\text{Cte } k) \ (\text{derivado } p))$
3. $\text{Num } a \Rightarrow \forall p :: \text{Polinomio } a. (\text{sinConstantesNegativas } p \implies \text{sinConstantesNegativas } (\text{derivado } p))$