

TP PLP

Javaneta

September 17, 2024

1 Ejercicio 9

De acuerdo a las definiciones de las funciones para arboles ternarios ($AT\ a$) se pide demostrar: $(\forall t::AT\ a)(\forall x::a)(elem\ x\ (preorder\ t) = elem\ x\ (postorder\ t))$

Preorder y Postorder son dos maneras diferentes de recorrer un árbol; En este caso en puntual, recorren un árbol ternario.

La lista resultante de ordenar el árbol ternario que devuelve preorder t y postorder t son diferentes, pero lo que ambos tienen en común es que contienen los mismos elementos, es decir, $(\forall x :: a)(x \in listaResPreOrder \iff x \in listaResPostOrder)$

Tenemos un caso que podemos mencionar, donde en el ejercicio 4 del TP: $elem\ n\ (preorder\ at) = elem\ n\ (postorder\ at)$ es válido $\forall n :: a$

Por lo tanto, probemos esto utilizando los principios de extensionalidad e inducción estructural sobre árboles para concluir que esto es verdadero para cualquier árbol ternario.

Recordemos la definición del tipo AT y cuáles son los constructores del tipo correspondientes: $data\ AT\ a = Nil \mid Tern\ a\ (AT\ a)\ (AT\ a)\ deriving\ Eq$

Luego, recordemos qué ecuaciones representan a las operaciones de $elem$, $preorder$ y $postorder$.

```
foldr :: (a -> b -> b) -> b -> [a] -> b
{FO0} foldr f z [] = z
{FO1} foldr f z (x:xs) = f x (foldr f z xs)
```

```
elem :: Eq a => a -> [a] -> Bool
{E0} elem e [] = False
{E1} elem e (x:xs) = (e==x) || elem e xs
```

```
foldAT :: (a -> b -> b -> b -> b) -> b -> AT a -> b
```

```

{F0} foldAT _ b Nil = b
{F1} foldAT f b (Tern r i c d) =
f r (foldAT f b i) (foldAT f b c) (foldAT f b d)

```

```

preorder :: AT a -> [a]
{PR0} preorder = foldAT(\r i c d -> r : (i ++ c ++ d)) []

```

```

postorder :: AT a -> [a]
{PS0} postorder = foldAT(\r i c d -> reverse (r : (d ++ c ++ i))) []

```

Por inducción estructural en t tenemos:

- $P(t) = (\forall x::a)(\text{elem } x (\text{preorder } t) = \text{elem } x (\text{postorder } t))$

Probemos el caso base, es decir, el constructor del tipo AT a no recursivo: este caso es Nil .

Caso Base: $P(Nil) = (\forall x::a)(\text{elem } x (\text{preorder } Nil) = \text{elem } x (\text{postorder } Nil))$
Resolvamos ambos lados por separado y deberíamos llegar a una equivalencia.

- $\text{elem } x (\text{preorder } Nil) \stackrel{PR0}{=} \text{elem } x (\text{foldAT}(\backslash r \ i \ c \ d \rightarrow r : (i ++ c ++ d)) [] Nil) \stackrel{F0}{=} \text{elem } x [] \stackrel{E0}{=} \text{False}$
- $\text{elem } x (\text{postorder } Nil) \stackrel{PS0}{=} \text{elem } x (\text{foldAT}(\backslash r \ i \ c \ d \rightarrow \text{reverse } (r : (d ++ c ++ i))) [] Nil) \stackrel{F0}{=} \text{elem } x [] \stackrel{E0}{=} \text{False}$

Por lo tanto, el caso base es verdadero.

Recordemos que para poder hacer inducción sobre listas necesitamos predicar acerca de $(\forall xs :: [a])(\forall x :: a)(P(xs) \implies P(x : xs))$. En árboles, la inducción lo hacemos sobre cada constructor recursivo, en este caso serían 3.

Paso Inductivo:

- $(\forall i, c, d :: AT \ a)(\forall r :: a)(P(i) \wedge P(c) \wedge P(d) \implies P(\text{Tern } r \ i \ c \ d))$
 - HI: $P(i) \wedge P(c) \wedge P(d)$ donde:
 - * $P(i)$: $\text{elem } x (\text{preorder } i) = \text{elem } x (\text{postorder } i)$
 - * $P(c)$: $\text{elem } x (\text{preorder } c) = \text{elem } x (\text{postorder } c)$
 - * $P(d)$: $\text{elem } x (\text{preorder } d) = \text{elem } x (\text{postorder } d)$
 - TI: $P(\text{Tern } r \ i \ c \ d)$ es decir
 - * $\text{elem } x (\text{preorder } (\text{Tern } r \ i \ c \ d)) = \text{elem } x (\text{postorder } (\text{Tern } r \ i \ c \ d))$

Por lo tanto

```

elem x (preorder (Tern r i c d))
 $\stackrel{PR0}{=}$  elem x (foldAT(\r i c d -> r : (i ++ c ++ d)) [] (Tern r i c d))
 $\stackrel{F1}{=}$  elem x (r : (foldAT (\r i c d -> r : (i ++ c ++ d)) [] i ++ foldAT (\r i c d
-> r : (i ++ c ++ d)) [] c ++ foldAT (\r i c d -> r : (i ++ c ++ d)) [] d))
 $\stackrel{PR0}{=}$  elem x (r : (preorder i) ++ (preorder c) ++ (preorder d))

```

$$\stackrel{E1}{=} (x==r) \parallel \text{elem } x ((\text{preorder } i) ++ (\text{preorder } c) ++ (\text{preorder } d))$$

Aplicamos el Lema 1

$$= (x==r) \parallel \text{elem } x (\text{preorder } i) \parallel \text{elem } x (\text{preorder } c) \parallel \text{elem } x (\text{preorder } d)$$

Por extensionalidad de booleanos en la condición de $(x==r)$ tenemos dos casos:

- A: $(x==r) = \text{True}$
- B: $(x==r) = \text{False}$

Caso A: $\text{True} \parallel \text{elem } x (\text{preorder } i) \parallel \text{elem } x (\text{preorder } c) \parallel \text{elem } x (\text{preorder } d) \rightarrow$, esto es verdadero ya que en el caso de un ó lógico basta un True para que todo sea verdadero.

Caso B: $\text{False} \parallel \text{elem } x (\text{preorder } i) \parallel \text{elem } x (\text{preorder } c) \parallel \text{elem } x (\text{preorder } d) \stackrel{HI}{=} \text{elem } x (\text{postorder } i) \parallel \text{elem } x (\text{postorder } c) \parallel \text{elem } x (\text{postorder } d) \stackrel{\text{Lema 1}}{=} \text{elem } x ((\text{postorder}(i)) ++ (\text{postorder } c) ++ (\text{postorder } d))$

Lema 1: Vamos a probar la siguiente propiedad sobre inducción estructural sobre xs.

$$(\forall xs :: [a])(\forall ys :: [a])(\forall e :: a)(\text{elem } e (xs ++ ys) = \text{elem } e xs \parallel \text{elem } e ys)$$

$$P(xs) = (\forall ys :: [a])(\forall e :: a)(\text{elem } e (xs ++ ys) = \text{elem } e xs \parallel \text{elem } e ys)$$

Caso Base: $P([])$

- $\text{elem } e ([] ++ ys) = \text{elem } e (\text{foldr}(:) \text{ ys } []) \stackrel{F0}{=} \text{elem } e ys$
- $\text{elem } e ([]) \parallel \text{elem } e (ys) \stackrel{E0}{=} \text{False} \parallel \text{elem } e ys = \text{elem } e ys$

Por lo tanto, el caso base es verdadero.

Paso Inductivo:

- HI: $P(xs): (\forall ys :: [a])(\forall e :: a)(\text{elem } e (xs ++ ys) = \text{elem } e xs \parallel \text{elem } e ys)$
- TI: $P(x:xs): (\forall ys :: [a])(\forall e :: a)(\text{elem } e ((x : xs) ++ ys) = \text{elem } e (x : xs) \parallel \text{elem } e ys)$

$$\text{elem } e ((x:xs) ++ ys) \stackrel{++}{=} \text{elem } e (\text{foldr}(:) \text{ ys } (x:xs)) \stackrel{FO1}{=} \text{elem } e ((:) x (\text{foldr}(:) \text{ ys } xs)) \stackrel{E1}{=} (e == x \parallel \text{elem } e (\text{foldr}(:) \text{ ys } xs))$$

Por el **principio de extensionalidad de booleanos** tengo dos casos para $e == x$

- LE1) $e == x = \text{True} = \text{True} \parallel \text{elem } e xs \parallel \text{elem } e ys = \text{True}$
- LE2) $e == x = \text{False} = \text{False} \parallel \text{elem } e xs \parallel \text{elem } e ys = \text{elem } e xs \parallel \text{elem } e ys$

El caso LE1 es trivial. El caso LE2 podemos aplicar la HI
 $\text{elem } e xs \parallel \text{elem } e ys \stackrel{HI}{=} \text{elem } e (xs ++ ys)$