

TP PLP

Javaneta

September 20, 2024

1 Ejercicio 9

Nota: El Lema utilizado está definido en la página 4.

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: $data\ AT\ a = Nil \mid Tern\ a\ (AT\ a)\ (AT\ a)\ (AT\ a)\ deriving\ Eq$

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 -> [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.

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

Como el lado izquierdo y el derecho de la expresión son equivalentes, 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
 - * $\text{elem } x (\text{preorder } (\text{Tern } r\ i\ c\ d)) = \text{elem } x (\text{postorder } (\text{Tern } r\ i\ c\ d))$

Lado Izquierdo: Para una mayor legibilidad, llamamos $\text{pf0} = \text{foldAT}(\backslash r\ i\ c\ d \rightarrow [r] ++ i ++ c ++ d)$
 $\text{elem } x (\text{preorder } (\text{Tern } r\ i\ c\ d))$
 $\stackrel{PR0}{=} \text{elem } x (\text{foldAT } (\backslash r\ i\ c\ d \rightarrow [r] ++ i ++ c ++ d) [] (\text{Tern } r\ i\ c\ d))$

$$\stackrel{F1}{=} \text{elem } x ([r] ++ (\text{foldAT } (\backslash r \text{ i c d } \rightarrow [r] ++ i ++ c ++ d) [] i) ++ (\text{foldAT } (\backslash r \text{ i c d } \rightarrow [r] ++ i ++ c ++ d) [] c) ++ (\text{foldAT } (\backslash r \text{ i c d } \rightarrow [r] ++ i ++ c ++ d) [] d))$$

$$\stackrel{PR0}{=} \text{elem } x ([r] ++ (\text{preorder } i) ++ (\text{preorder } c) ++ (\text{preorder } d))$$

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

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: $(x == r) = \text{False}$

$$\textbf{Caso A: } \text{True} \parallel \text{elem } x (\text{preorder } i) \parallel \text{elem } x (\text{preorder } c) \parallel \text{elem } x (\text{preorder } d) = \text{True}$$

Esto es verdadero ya que en el caso de un ó lógico basta un True para que todo sea verdadero.

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

Veamos ahora, que si desarrollamos desde **elem x (postorder(Tern r i c d))** y llegamos a aplicar la hipótesis inductiva como esto es una igualdad, basta para que sea verdadero.

Lado Derecho

$$\text{elem } x (\text{postorder}(\text{Tern } r \text{ i c d}))$$

$$\stackrel{PS0}{=} \text{elem } x (\text{foldAT}(\backslash r \text{ i c d } \rightarrow [r] ++ d ++ c ++ i) [] (\text{Tern } r \text{ i c d}))$$

$$\stackrel{F1}{=} \text{elem } x ([r] ++ (\text{foldAT}(\backslash r \text{ i c d } \rightarrow [r] ++ d ++ c ++ i) [] i) ++ (\text{foldAT}(\backslash r \text{ i c d } \rightarrow [r] ++ d ++ c ++ i) [] c) ++ (\text{foldAT}(\backslash r \text{ i c d } \rightarrow [r] ++ d ++ c ++ i) [] d))$$

$$\stackrel{PS0}{=} \text{elem } x ([r] ++ \text{postorder } i ++ \text{postorder } c ++ \text{postorder } d)$$

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

Nuevamente por Lema 1

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

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

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

$$\textbf{Caso A: } \text{True} \parallel \text{elem } x (\text{postorder } i) \parallel \text{elem } x (\text{postorder } c) \parallel \text{elem } x (\text{postorder } d) = \text{True}$$

$$\textbf{Caso B: } \text{False} \parallel \text{elem } x (\text{postorder } i) \parallel \text{elem } x (\text{postorder } c) \parallel \text{elem } x (\text{postorder } d)$$

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

Luego, como probamos que de ambos lados pudimos aplicar la HI, y la HI es verdadera, entonces sucede que: $\text{elem } x \text{ (preorder (Tern } r \text{ i } c \text{ d))} = \text{elem } x \text{ (postorder (Tern } r \text{ i } c \text{ d))}$ como queríamos probar.

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

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

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

Caso Base: $P([])$

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

Por lo tanto, el caso base es verdadero.

Paso Inductivo:

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

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

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

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

Veamos ahora por el otro lado

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

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

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

Luego, queda probado el Lema.