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: (∀t::AT a)(∀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: $data\ AT\ a\ =\ Nil\ |\ Tern\ a\ (AT\ a)\ (AT\ a)\ (AT\ a)\ deriving\ Eq$

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 -> reverse (r : (d ++ c ++ i))) []
```

Por inducción estructural en t tenemos:

- P(t) = (∀ x::a)(elem x (preorder t) = elem x (postorder t))

Probemos el caso base, es decir, el constructor del tipo AT a no recursivo: este caso es Nil.
Caso Base: P(Nil) = (∀ x::a)(elem x (preorder Nil) = elem x (postorder Nil))
Resolvamos ambos lados por separado y deberíamos llegar a una equivalencia.

- elem x (preorder Nil) $\overset{\text{PR0}}{=}$ elem x (foldAT(\ r i c d → r : (i ++ c ++ d)) [] Nil) $\overset{\text{F0}}{=}$ elem x [] $\overset{\text{E0}}{=}$ False

- elem x (postorder Nil) $\overset{\text{PS0}}{=}$ elem x (foldAT(\ r i c d → reverse (r : (d ++ c ++ i))) [] Nil) $\overset{\text{F0}}{=}$ elem x [] $\overset{\text{E0}}{=}$ False

Como el lado iszquierdo 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:

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

Por lo tanto
elem x (preorder (Tern r i c d))
$\overset{\text{PR0}}{=}$ elem x (foldAT(\r i c d → r : (i ++ c ++ d)) [] (Tern r i c d))
$\overset{\text{F1}}{=}$ 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))

$\stackrel{\text{PR0}}{=}$ elem x (r : (preorder i) ++ (preorder c) ++ (preorder d))
$\stackrel{\text{E1}}{=}$ (x==r) ‖ elem x ((preorder i) ++ (preorder c) ++ (preorder d))
Aplicamos el Lema 1
= (x==r) ‖ elem x (preorder i) ‖ elem x (preorder c) ‖ elem x (preorder d)
Por extensionalidad de booleanos en la condición de (x==r) tenemos dos casos:

- A: (x==r) = True
- B: (x==r) = False

Caso A: True ‖ elem x (preorder i) ‖ elem x (preorder c) ‖ elem x (preorder d)
= True
Esto es verdadero ya que en el caso de un ó lógico basta un True para que todo sea verdadero.

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

Lema 1: Vamos a probar la siguiente propiedad sobre inducción estructural sobre xs.
$(\forall xs :: [a])(\forall ys :: [a])(\forall e :: a)(elem\ e\ (xs++ys)\ = elem\ e\ xs \parallel elem\ e\ ys$
P(xs) = $(\forall ys :: [a])(\forall e :: a)(elem\ e\ (xs++ys)\ = elem\ e\ xs \parallel elem\ e\ ys)$
Caso Base: P([])

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

Por lo tanto, el caso base es verdadero.
Paso Inductivo:

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

elem e ((x:xs) ++ ys) $\stackrel{++}{=}$ elem e (foldr (:) ys (x:xs)) $\stackrel{\text{FO1}}{=}$ elem e ((:) x (foldr(:) ys xs) $\stackrel{\text{E1}}{=}$ (e == x ‖ elem e (foldr(:) ys xs)
Por el **principio de extensionalidad de booleanos** tengo dos casos para $e == x$

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

Veamos ahora por el otro lado
elem e (x:xs) $\|$ elem ys $\overset{\text{E1}}{=}$ (e == x $\|$ elem e xs $\|$ elem e ys)
Por el **principio de extensionalidad de booleanos** tengo dos casos para $e == x$

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

Luego, queda probado el Lema.