TP1 - Programación Funcional

Pollo al verdeo

September 2024

1 Enunciado

Probar que:
$\forall$ t :: AT a . $\forall$ x :: a . (elem x (preorder t) = elem x (postorder t))

2 Definiciones

2.1 Elem

elem :: Eq a $\Rightarrow$ a $\rightarrow$ [a] $\rightarrow$ Bool

1. E0: elem e [] = False
2. E1: elem e (x:xs) = (e ==x) || elem e xs

2.2 (++)

(++) :: [a] $\rightarrow$ [a] $\rightarrow$ [a]

1. C0: (++) [] ys = ys
2. C1: (++) (x:xs) ys = x : (xs ++ ys)

2.3 FoldAT

foldAT :: (a $\rightarrow$ b $\rightarrow$ b $\rightarrow$ b $\rightarrow$ b) $\rightarrow$ b $\rightarrow$ AT a $\rightarrow$ b

1. F0: foldAT cTern cNil Nil = cNil
2. F1: foldAT cTern cNil (Tern raiz izq cen der) =
   cTern raiz (f izq) (f cen) (f der)
   where f = foldAT cTern cNil

2.4 Preorder

preorder :: Procesador (AT a) a

1. PE0: preorder t = foldAT g [] t
2. g: (\raiz izq cen der $\rightarrow$ [raiz] ++ izq ++ cen ++ der)

2.5 Postorder

postorder :: Procesador (AT a) a

1. PO0: postorder t = foldAT h [] t
2. h: (\raiz izq cen der → izq ++ cen ++ der ++ [raiz])

2.6 Árbol ternario

data AT a = Nil || Tern a (AT a) (AT a) (AT a) deriving Eq

3 Lema

Para la prueba por inducción vamos a necesitar probar primero que un elemento pertenece a la concatenación de dos listas si y solo si el elemento pertenece a alguna de las listas que la conforman:

elem e (s1 ++ s2) = elem e s1 || elem e s2

Vamos a probar este lema por inducción sobre la primera de las dos listas a concatenar (s1), y manteniendo fija la otra, (s2).

3.1 Caso base

Veamos si se cumple el caso base, cuando la primera lista es vacía.

3.1.1 Caso s1 lista vacía, lado izquierdo

elem e (s1 ++ s2)
$=_{s1}$ elem e ([] ++ s2)
$=_{C0}$ elem e s2

3.1.2 Caso s1 lista vacía, lado derecho

elem e s1 || elem e s2
$=_{s1}$ elem e [] || elem e s2
$=_{E0}$ False || elem e s2
$=_{Bool}$ elem e s2

Se cumple el caso base.

3.2 Caso inductivo

Veamos ahora el caso inductivo sobre la lista s1, no vacía. Nuestra hipótesis inductiva es:

elem e (xs ++ s2) = elem e xs || elem e s2

Y la tesis inductiva es:

elem e ((x:xs) ++ s2) = elem e (x:xs) || elem e s2

3.2.1 Caso s1 una lista no vacía, lado izquierdo

elem e (s1 ++ s2)
$=_{s1}$ elem e ((x:xs) ++ s2)
$=_{C1}$ elem e (x : (xs ++ s2))
$=_{E1}$ e == x || elem e (xs ++ s2)
$=_{HI}$ e == x || elem e xs || elem e s2
$=_{E1}$ elem e (x:xs) || elem e (s2)
$=_{s1}$ elem e s1 || elem e s2

Se cumple el caso inductivo, llegamos a nuestro lado derecho.

4 Prueba

Queremos ver que ∀ t :: AT a . ∀ x :: a . (elem x (preorder t) = elem x (postorder t)). Vamos a realizar una prueba inductiva sobre t, nuestro árbol ternario.

4.1 Caso base

Veamos si se cumple para nuestro caso base, un árbol ternario de tipo "a" formado por el constructor Nil.

4.1.1 Caso Nil, lado izquierdo

elem x (preorder Nil)
$=_{PE0}$ elem x (foldAT g [] Nil) where g = (\raiz izq cen der → [raiz] ++ izq ++ cen ++ der)
$=_{F0}$ elem x ([])
$=_{E0}$ False

4.1.2 Caso Nil, lado derecho

elem x (postorder Nil)
$=_{PO0}$ elem x (foldAT h [] Nil) where h = (\raiz izq cen der → izq ++ cen ++ der ++ [raiz])
$=_{F0}$ elem x ([])
$=_{E0}$ False

Se cumple el caso base.

4.2 Caso inductivo

Veamos si se cumple el caso inductivo, un árbol ternario de tipo "a" de la forma Tern a (AT a) (AT a) (AT a).
Tenemos que nuestro P(t) es:
∀ t :: AT a . ∀ x :: a . (elem x (preorder t) = elem x (postorder t)).
Al querer probarlo para un árbol ternario, nuestra hipótesis inductiva es que esto

debe cumplirse para las ramas que forman al árbol (centro, derecha e izquierda).

H.I. = P(izq) $\wedge$ P(cen) $\wedge$ P(der)

4.2.1 Caso t un árbol ternario no Nil, lado izquierdo

elem x (preorder (Tern raiz izq cen der))
$=_{PE0}$ elem x (foldAT g [] (Tern raiz izq cen der))
$=_{F1}$ elem x (g raiz (foldAT g [] izq) (foldAT g [] cen) (foldAT g [] der))
$=_{PE0}$ elem x (g raiz (preorder izq) (preorder cen) (preorder der))
$=_{g}$ elem x ([raiz] ++ (preorder izq) ++ (preorder cen) ++ (preorder der))
$=_{lema}$ elem x ([raiz] ++ (preorder izq) ++ (preorder cen)) || elem x (preorder der)
$=_{lema}$ elem x ([raiz] ++ (preorder izq)) || elem x (preorder cen)) || elem x (preorder der)
$=_{lema}$ elem x [raiz] || elem x (preorder izq) || elem x (preorder cen) || elem x (preorder der)
$=_{HI}$ elem x [raiz] || elem x (postorder izq) || elem x (postorder cen) || elem x (postorder der)
$=_{bool}$ elem x (postorder izq) || elem x (postorder cen) || elem x (postorder der) || elem x [raiz]
$=_{lema}$ elem x ((postorder izq) ++ (postorder cen)) || elem x (postorder der) || elem x [raiz])
$=_{lema}$ elem x ((postorder izq) ++ (postorder cen) ++ (postorder der)) || elem x [raiz])
$=_{lema}$ elem x ((postorder izq) ++ (postorder cen) ++ (postorder der) ++ [raiz])
$=_{h}$ elem x (h raiz (postorder izq) (postorder cen) (postorder der))
$=_{PO0}$ elem x (h raiz (foldAT h [] izq) (foldAT h [] cen) (foldAT h [] der))
$=_{F1}$ elem x (foldAT h [] (Tern raiz izq cen der))
$=_{PO0}$ elem x (postorder t)

Se cumple el caso inductivo, llegamos al lado derecho.
Queda demostrado entonces, que $\forall$ t :: AT a . $\forall$ x :: a . (elem x (preorder t) = elem x (postorder t))