

# TP PLP

Javaneta

September 15, 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  $postorder$ .

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

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

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:**

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

Por lo tanto

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

$\stackrel{\text{PR0}}{=} \text{elem } x (\text{foldAT}(\backslash r \text{ i c d} \rightarrow r : (i ++ c ++ d)) [] (\text{Tern } r \text{ i c d}))$

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

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

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

¿Alguna propiedad concatenación?

$= (x==r) \parallel \text{elem } x (\text{preorder } i) \parallel \text{elem } x (\text{preorder } c) \parallel \text{elem } x (\text{preorder } d)$  A simple vista, en este momento podríamos aplicar la HI. 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:  $\text{True} \parallel \text{elem } x \text{ (preorder i)} \parallel \text{elem } x \text{ (preorder c)} \parallel \text{elem } x \text{ (preorder d)} \stackrel{\text{HI}}{=} \text{elem } x \text{ (postorder i)} \parallel \text{elem } x \text{ (postorder c)} \parallel \text{elem } x \text{ (postorder d)} \rightarrow$  Creo que acá ya sería una vuelta atrás y llegaríamos. Pero tengo dudas sobre el momento de la concatenación

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