

Haskell, el lenguaje de los perezosos

Marcos Viera

Instituto de Computación, Universidad de la República
Montevideo, Uruguay

ScaLATAM - 02/05/2019

Algunas de la principales características de Haskell

- Lenguaje Funcional Puro

Algunas de la principales características de Haskell

- Lenguaje Funcional Puro
- Sintaxis Minimal

Algunas de la principales características de Haskell

- Lenguaje Funcional Puro
- Sintaxis Minimal
- Tipado Estático e Inferencia de Tipos

Algunas de la principales características de Haskell

- Lenguaje Funcional Puro
- Sintaxis Minimal
- Tipado Estático e Inferencia de Tipos
- Alto Orden

Algunas de la principales características de Haskell

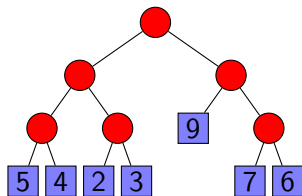
- Lenguaje Funcional Puro
- Sintaxis Minimal
- Tipado Estático e Inferencia de Tipos
- Alto Orden
- Evaluación Perezosa

El *repm* problem

Dado un árbol binario con valores en las hojas, computar un nuevo árbol reemplazando los valores de sus hojas por su mínimo.

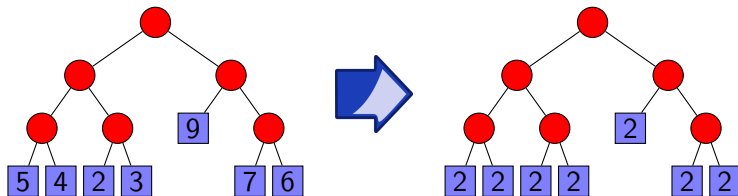
El *repmin* problem

Dado un árbol binario con valores en las hojas, computar un nuevo árbol reemplazando los valores de sus hojas por su mínimo.



El *repm* problem

Dado un árbol binario con valores en las hojas, computar un nuevo árbol reemplazando los valores de sus hojas por su mínimo.



Tipos Algebraicos

Dada la definición

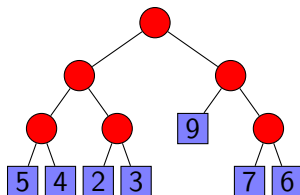
```
data Tree a = Node Tree Tree  
            | Leaf a
```

Tipos Algebraicos

Dada la definición

```
data Tree a = Node Tree Tree  
             | Leaf a
```

El siguiente árbol



se puede representar con un valor de tipo *Tree*

```
t = Node (Node (Node (Leaf 5) (Leaf 4))  
          (Node (Leaf 2) (Leaf 3)))  
      (Node (Leaf 9)  
            (Node (Leaf 7) (Leaf 6)))
```

Una primera solución

Definimos funciones para computar el mínimo y para reemplazar las hojas por un valor

```
minTree :: Ord a ⇒ Tree a → a
```

```
minTree (Leaf x) = x
```

```
minTree (Node l r) = min (minTree l) (minTree r)
```

```
replace :: Tree a → a → Tree a
```

```
replace (Leaf _) x = Leaf x
```

```
replace (Node l r) x = Node (replace l x) (replace r x)
```

Una primera solución

Definimos funciones para computar el mínimo y para reemplazar las hojas por un valor

```
minTree :: Ord a => Tree a -> a  
minTree (Leaf x)    = x  
minTree (Node l r) = min (minTree l) (minTree r)  
  
replace :: Tree a -> a -> Tree a  
replace (Leaf _)   x = Leaf x  
replace (Node l r) x = Node (replace l x) (replace r x)
```

Entonces *repmin* sería

```
repmin :: Ord a => Tree a -> Tree a  
repmin t = replace t (minTree t)
```

Una primera solución

Definimos funciones para computar el mínimo y para reemplazar las hojas por un valor

```
minTree :: Ord a => Tree a -> a
minTree (Leaf x)    = x
minTree (Node l r) = min (minTree l) (minTree r)

replace :: Tree a -> a -> Tree a
replace (Leaf _)   x = Leaf x
replace (Node l r) x = Node (replace l x) (replace r x)
```

Entonces *repmin* sería

```
repmin :: Ord a => Tree a -> Tree a
repmin t = (replace t . minTree) t
```

Una primera solución

Definimos funciones para computar el mínimo y para reemplazar las hojas por un valor

```
minTree (Leaf x)    = x  
minTree (Node l r) = min (minTree l) (minTree r)
```

```
replace (Leaf _)    x = Leaf x  
replace (Node l r) x = Node (replace l x) (replace r x)
```

Entonces *repmin* sería

```
repmin t = (replace t . minTree) t
```

Las anotaciones de tipos no son necesarias.

Abstrayendo Patrones Comunes

- Si definimos una función para representar el patrón de recursión

```
cata :: (a → b) → (b → b → b) → Tree a → b  
cata fl _ (Leaf x)    = fl x  
cata fl fn (Node l r) = fn (cata fl fn l) (cata fl fn r)
```


Abstrayendo Patrones Comunes

- Si definimos una función para representar el patrón de recursión

```
cata :: (a → b) → (b → b → b) → Tree a → b  
cata fl _ (Leaf x) = fl x  
cata fl fn (Node l r) = fn (cata fl fn l) (cata fl fn r)
```

Entonces

```
minTree t = cata id min t  
replace t x = cata (Leaf . const x) Node t
```

Abstrayendo Patrones Comunes

- Si definimos una función para representar el patrón de recursión

```
cata :: (a → b) → (b → b → b) → Tree a → b  
cata fl _ (Leaf x) = fl x  
cata fl fn (Node l r) = fn (cata fl fn l) (cata fl fn r)
```

Entonces

```
minTree t = cata id min t  
replace t x = cata (Leaf . const x) Node t
```

- Si definimos una función para representar el mapeo en el árbol

```
mapTree :: (a → b) → Tree a → Tree b  
mapTree f = cata (Leaf . f) Node
```

Abstrayendo Patrones Comunes

- Si definimos una función para representar el patrón de recursión

```
cata :: (a → b) → (b → b → b) → Tree a → b  
cata fl _ (Leaf x) = fl x  
cata fl fn (Node l r) = fn (cata fl fn l) (cata fl fn r)
```

Entonces

```
minTree t = cata id min t  
replace t x = cata (Leaf . const x) Node t
```

- Si definimos una función para representar el mapeo en el árbol

```
mapTree :: (a → b) → Tree a → Tree b  
mapTree f = cata (Leaf . f) Node
```

Entonces

```
replace t x = mapTree (const x) t
```

Otra solución que usa Alto Orden

Durante la recorrida computar el mínimo

```
repmin' (Leaf x) = (x)
repmin' (Node l r) = let ( ml) = repmin' l
                        (  mr) = repmin' r
                        in ( min ml mr)
```

Otra solución que usa Alto Orden

Durante la recorrida computar el mínimo y la función que reemplaza el valor en el árbol

```
repmin' (Leaf x) = (λx → Leaf x, )  
repmin' (Node l r) = let (fl, ) = repmin' l  
                        (fr, ) = repmin' r  
                        in (λx → Node (fl x) (fr x), )
```

Otra solución que usa Alto Orden

Durante la recorrida computar el mínimo y la función que reemplaza el valor en el árbol, juntos.

```
repmin' (Leaf x)  = (λx → Leaf x, x)
repmin' (Node l r) = let (fl, ml) = repmin' l
                      (fr, mr) = repmin' r
                      in (λx → Node (fl x) (fr x), min ml mr)
```

Otra solución que usa Alto Orden

Durante la recorrida computar el mínimo y la función que reemplaza el valor en el árbol, juntos.

```
repmin' :: Ord a => Tree a -> (b -> Tree b, a)
repmin' (Leaf x)   = (\x -> Leaf x, x)
repmin' (Node l r) = let (fl, ml) = repmin' l
                        (fr, mr) = repmin' r
                        in (\x -> Node (fl x) (fr x), min ml mr)
```

Otra solución que usa Alto Orden

Durante la recorrida computar el mínimo y la función que reemplaza el valor en el árbol, juntos.

```
repmin' :: Ord a => Tree a -> (b -> Tree b, a)
repmin' (Leaf x)   = (\x -> Leaf x, x)
repmin' (Node l r) = let (fl, ml) = repmin' l
                        (fr, mr) = repmin' r
                        in (\x -> Node (fl x) (fr x), min ml mr)
```

Entonces *repmin* sería

```
repmin t = let (f, m) = repmin' t
            in f m
```


Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5
```

```
= 5 : repeat 5
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5  
= 5 : repeat 5  
= 5 : 5 : repeat 5
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5  
= 5 : repeat 5  
= 5 : 5 : repeat 5  
...  
= 5:5:5:5:5:...
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5  
= 5 : repeat 5  
= 5 : 5 : repeat 5  
...  
= 5:5:5:5:5:...  
  
> head (repeat 5)
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5  
= 5 : repeat 5  
= 5 : 5 : repeat 5  
...  
= 5:5:5:5:5:...  
  
> head (repeat 5)  
= head (5 : repeat 5)
```

Si tenemos

```
repeat x = x : repeat x
```

```
head (x: _) = x
```

```
> repeat 5  
= 5 : repeat 5  
= 5 : 5 : repeat 5  
...  
= 5:5:5:5:5:...  
  
> head (repeat 5)  
= head (5 : repeat 5)  
= 5
```


Solución de una sola recorrida, usando Evaluación Perezosa

Durante la recorrida computar el mínimo y reemplazar por un valor dado.

```
repmin' (Leaf x) m = (Leaf m, x)
repmin' (Node l r) m = let (l', ml) = repmin' l m
                        (r', mr) = repmin' r m
                        in (Node l' r', min ml mr)
```

Solución de una sola recorrida, usando Evaluación Perezosa

Durante la recorrida computar el mínimo y reemplazar por un valor dado.

```
repmin' (Leaf x) m = (Leaf m, x)
repmin' (Node l r) m = let (l', ml) = repmin' l m
                        (r', mr) = repmin' r m
                        in (Node l' r', min ml mr)
```

Podemos definir *repmin*

```
repmin t = let (t', m) = repmin' t m
            in t'
```


Solución de una sola recorrida, usando Evaluación Perezosa

Durante la recorrida computar el mínimo y reemplazar por un valor dado.

```
repmin' (Leaf x) m = (Leaf m, x)
repmin' (Node l r) m = let (l', ml) = repmin' l m
                        (r', mr) = repmin' r m
                        in (Node l' r', min ml mr)
```

Podemos definir *repmin*

```
repmin t = let (t', m) = repmin' t m
            in t'
```



usando *programación circular*

Muchas Gracias!