Facultad de Ciencias, UNAM
Programación Declarativa
Tarea 3: Bringing you into the fold

Rubí Rojas Tania Michelle

12 de marzo de 2020

1. Demuestra que las siguientes propiedades de los operadores de plegado:

   a) foldr f e . map g = foldr (f . g) e

   Demostración. Inducción estructural sobre xs.

   - Caso base.
     $xs = []$. Este caso se cumple ya que

$$
\begin{aligned}
(foldr\ f\ e\ .\ map\ g)\ [] &= foldr\ f\ e\ (map\ g\ []) && \\
&= foldr\ f\ e\ [] && \text{def. de map} \\
&= e && \text{def. de foldr} \\
&= foldr\ (f\ .\ g)\ e\ [] &&
\end{aligned}
$$

   - Hipótesis de inducción.

     foldr f e . map g = foldr (f . g) e

   - Paso inductivo.

$$
\begin{aligned}
(foldr\ f\ e\ .\ map\ g)\ (x : xs) &= foldr\ f\ e\ (map\ g\ (x : xs)) && \\
&= foldr\ f\ e\ (g\ x\ :\ map\ g\ xs) && \text{def. de map} \\
&= f\ (g\ x)\ (foldr\ f\ e\ (map\ g\ xs)) && \text{def. de foldr} \\
&= f\ (g\ x)\ (foldr\ (f\ .\ g)\ e\ xs) && \text{hipótesis de inducción} \\
&= (f\ .\ g)\ x\ (foldr\ (f\ .\ g)\ e\ xs) && \text{def. de composición} \\
&= foldr\ (f\ .\ g)\ e\ (x : xs) && \text{def. de foldr}
\end{aligned}
$$

   □

   b) foldl f e xs = foldr (flip f) e (reverse xs)

   Demostración. Inducción estructural sobre xs.

   - Caso base.
     $xs = []$. Este caso se cumple ya que

$$
\begin{aligned}
foldl\ f\ e\ [] &= e && \text{def. de foldl} \\
&= foldr\ (flip\ f)\ e\ [] && \text{def. de foldr} \\
&= foldr\ (flip\ f)\ e\ (reverse\ []) && \text{def. de reverse}
\end{aligned}
$$

- Hipótesis de inducción.

foldl f e xs = foldr (flip f) e (reverse xs)

- Paso inductivo.

$$
\begin{aligned}
foldr\ (flip\ f)\ e\ (reverse\ (x:xs)) &= foldr\ (flip\ f)\ e\ (reverse\ xs ++[x]) \\
&= foldr\ (flip\ f)\ (foldr\ (flip\ f)\ e\ [x])\ (reverse\ xs) \\
&= foldr\ (flip\ f)\ ((flip\ f)\ [x]\ (foldr\ (flip\ f)\ e\ []))\ (reverse\ xs) \\
&= foldr\ (flip\ f)\ ((flip\ f)\ [x]\ e)\ (reverse\ xs) \\
&= foldr\ (flip\ f)\ (f\ e\ x)\ (reverse\ xs) \\
&= foldl\ f\ (f\ e\ x)\ xs \\
&= foldl\ f\ e\ (x:xs)
\end{aligned}
$$

donde la justificación de los pasos es la siguiente:

a) Aplicamos la definición de reverse.
b) Aplicamos lo demostrado en el inciso c).
c) Aplicamos la definición de foldr en (foldr (flip f) e [x]).
d) Aplicamos la definición de foldr en (foldr (flip f) e [])).
e) Aplicamos la definición de flip.
f) Aplicamos la hipótesis de inducción.
g) Aplicamos la definición de foldl.

□

c) foldr f e (xs ++ ys) = foldr f (foldr f e ys) xs

Demostración. Inducción estructural sobre xs.

- Caso base.
  $xs = []$. Este caso se cumple ya que

$$
\begin{aligned}
foldr\ f\ e\ ([]\ ++\ ys) &= foldr\ f\ e\ ys && \text{def. de ++} \\
&= foldr\ f\ (foldr\ f\ e\ ys)\ [] && \text{def. de foldr}
\end{aligned}
$$

- Hipótesis de inducción.

foldr f e (xs ++ ys) = foldr f (foldr f e ys) xs

- Paso inductivo.

$$
\begin{aligned}
foldr\ f\ e\ ((x:xs)\ ++\ ys) &= foldr\ f\ e\ (x:(xs++ys)) && \text{def. de ++} \\
&= f\ x\ (foldr\ f\ e\ (xs++ys)) && \text{def. de foldr} \\
&= f\ x\ (foldr\ f\ (foldr\ f\ e\ ys)\ xs) && \text{hipótesis de inducción} \\
&= foldr\ f\ (foldr\ f\ e\ ys)\ (x:xs) && \text{def. de foldr}
\end{aligned}
$$

□

2. Considera el siguiente tipo de dato algebraico en Haskell para definir árboles binarios.

```haskell
data Tree a = Void | Node (Tree a) a (Tree a)

```

Y la función *foldT* que define el operador de plegado para la estructura Tree, definido como sigue:

```haskell
foldT :: (b -> a -> b -> b) -> b -> Tree a -> b
foldT _ v Void = v
foldT f v (Node t1 r t2) = f t1' r t2'
  where t1' = foldT f v t1
        t2' = foldT f v t2
```

a) Da en términos de una función h el patrón encapsulado por el operador *foldT*.

Solución: Queremos resolver la ecuación $foldT\ f\ v = foldr\ h\ b$. Por la Propiedad Universal de Fold, esta ecuación es equivalente a

```haskell
foldT f v Void = b
foldT f v (Node t1 r t2) = h (foldT f v t1) r (foldT f v t2)
```

De la primera ecuación tenemos que $b = v$ por la definición de $foldT$. De la segunda ecuación, calculamos a h como sigue:

$$\begin{gathered}
\text{foldT f v (Node t1 r t2)} = \text{h (foldT f v t1) r (foldT f v t2)} \\
\Leftrightarrow \text{ definición de } foldT \\
\text{f (foldT f v t1) r (foldT f v t2)} = \text{h (foldT f v t1) r (foldT f v t2)} \\
\Leftrightarrow \text{ generalizando } \textit{(foldT f v t1) r (foldT f v t2)} \text{ como } tree \\
\text{f tree} = \text{h tree} \\
\Leftrightarrow \text{ por extensionalidad de funciones} \\
\text{f} = \text{h}
\end{gathered}$$

Por lo tanto, el patrón encapsulado por el operador $foldT$ es

```haskell
foldT f v = foldr f v
```

b) Enuncia y demuestra la propiedad Universal del operador *foldT*, basándote en la Propiedad Universal vista en clase sobre el operador *foldr* de listas.

Demostración. La Propiedad Universal del operador $foldT$ es

```
foldT f v Void = v                            ⇔ foldT f v = foldr f v
foldT f v (Node t1 r t2) = f t1' r t2'
  where t1' = foldT f r t1
        t2' = foldT f r t2
```

Para demostrar la siguiente igualdad

$$\text{foldT f v} = \text{foldr f v}$$

Como la igualdad es una instancia de la consecuencia de la propiedad universal, basta con demostrar las precondiciones de ésta, es decir,

```haskell
foldT f v Void = v
foldT f v (Node t1 r t2) = f (foldT f v t1) r (foldT f v t2)
```

que pueden ser verificadas fácilmente

- $Tree = Void$. Se cumple ya que

$$foldT\ f\ v\ Void = v \qquad\qquad \text{def. de foldT}$$

- $Tree = (Node\ t1\ r\ t2)$. Se cumple ya que

$$\begin{aligned} foldT\ f\ v\ (Node\ t1\ r\ t2) &= f\ t1'\ r\ t2' && \text{definición de foldT} \\ &= f\ (foldT\ f\ v\ t1)\ r\ (foldT\ f\ v\ t2) && \text{definición de } t1' \text{ y } t2' \end{aligned}$$

con lo que quedan demostradas las dos precondiciones. Aplicando la propiedad universal queda demostrada la propiedad original.

□

3. Calcula una definición eficiente para *scanr* partiendo de la siguiente:

$$\text{scanr f e} = \text{map (foldr f e) . tails}$$

SOLUCIÓN: Calculamos la definición de la siguiente manera

- $xs = []$. Tenemos que

$$\begin{aligned} scanr\ f\ e\ [] &= map\ (foldr\ f\ e)\ (tails\ []) \\ &= map\ (foldr\ f\ e)\ [[]] && \text{def. de tails} \\ &= map\ (foldr\ f\ e)([] : []) && \text{def. de :} \\ &= foldr\ f\ e\ [] : map\ (foldr\ f\ e)\ [] && \text{def. de map} \\ &= e : map\ (foldr\ f\ e)\ [] && \text{def. de foldr} \\ &= e : [] && \text{def. de map} \\ &= [e] && \text{def. de :} \end{aligned}$$

- $xs = (y : ys)$. Tenemos que

$$\begin{aligned} scanr\ f\ e\ (y : ys) &= map\ (foldr\ f\ e)\ (tails\ (y : ys)) && \text{especificación de scanr} \\ &= map\ (foldr\ f\ e)\ ((y : ys) : tails\ ys) && \text{def. de tails} \\ &= foldr\ f\ e\ (y : ys) : map\ (foldr\ f\ e)\ (tails\ ys) && \text{def. de map} \\ &= f\ y\ (foldr\ f\ e\ ys) : map\ (foldr\ f\ e\ (tails\ ys)) && \text{def. de foldr} \\ &= f\ y\ (foldr\ f\ e\ ys) : scanr\ f\ e\ ys && \text{especificación de scanr} \\ &= f\ y\ (head\ (scanr\ f\ e\ ys)) : scanr\ f\ e\ ys && \text{head (scanr f z xs) == foldr f z xs.} \end{aligned}$$

La última igualdad la obtuve de https://hackage.haskell.org/package/base-4.12.0.0/docs/GHC-List.html

Por lo tanto, la definición más eficiente de *scanr* sería

```
        scanr f e [] = [e]
        scanr f e (x:xs) = f x (head ys) : ys
          where ys = scanr f e xs

```

La especificación original requiere $O(n^2)$ aplicaciones de una lista de longitud n, mientras que esta nueva definición sólo usa $O(n)$ aplicaciones para una lista de longitud n.

4. Considera la siguiente definición de la función *cp* que calcula el producto cartesiano.

```
         cp :: [[a]] -> [[a]]
         cp = foldr f e

```

a) En la definición anterior, ¿quiénes son f y e?

SOLUCIÓN: La función *cp* calcula el producto cartesiano de una lista de listas, por lo que f sería de la forma

$$\text{f xs xss} = [(\text{x:ys}) \mid \text{x} \leftarrow \text{xs, ys} \leftarrow \text{xss}]$$

y el elemento e es igual a [[]].

b) Dada la siguiente ecuación

$$\text{length . cp} = \text{product . map length}$$

en donde *length* calcula la longitud de una lista y *product* regresa el resultado de la multiplicación de todos los elementos de una lista. Demuestra que la ecuación es cierta, para esto es necesario reescribir ambos lado de la ecuación como instancias de *foldr* y ver que son idénticas.

SOLUCIÓN: Debemos reescribir ambos lados de la ecuación como instancias de *foldr*. El lado izquierdo de la expresión lo queremos expresar como

```
            length . cp = foldr h b
```

Por la definición de *cp*, esta expresión se puede reescribir como

$$(\text{length . foldr f [[]] } \textbf{where} \text{ f xs xss} = [(\text{x:ys}) \mid \text{x} \leftarrow \text{xs, ys} \leftarrow \text{xss}]) = \text{foldr h b}$$

Por el teorema de fusión basta que se cumplan las siguientes dos condiciones (sabemos que *length* es una función estricta):

```
            length [[]]                         = b
            (length (f zs zss) where f xs xss = [(x:ys) | x <- xs, ys <- xss])
                                                = h zs (length zss)
```

De la primera ecuación obtenemos que $b = 1$ por la definición de *length*. La segunda ecuación puede transformarse en otra condición más restrictiva, de la cual obtenemos

$$\begin{gathered}
\text{length (f zs zss)} = \text{h zs (length zss)} \\
\Leftrightarrow |S \times T| = |S| \times |T| \\
\text{length zs} * \text{length zss} = \text{h zs (length zss)} \\
\Leftrightarrow \text{generalizando } \textit{length zss} \text{ como } ws \\
\text{length zs} * \text{ws} = \text{h zs ws} \\
\Leftrightarrow \text{por extensionalidad de funciones} \\
(*) \text{ (length zs)} = \text{h zs} \\
\Leftrightarrow \text{def. de composición de funciones} \\
((*) \text{ . length) zs} = \text{h zs} \\
\Leftrightarrow \text{por extensionalidad de funciones} \\
((*) \text{ . length)} = \text{h}
\end{gathered}$$

Por lo tanto,

```
            length . cp = foldr ((*) . length) 1
```

Ahora bien, el lado derecho de la expresión lo queremos expresar como

```
            product . map length = foldr h b
```

Primero, expresaremos a *map length* como una instancia de *foldr*. Sabemos que la función *map* puede escribirse como

$$\text{map f} = (\lambda\text{y ys} \rightarrow \text{(f y) : ys}) \; []$$

de donde

$$\begin{aligned}
\lambda y \; ys \; \rightarrow \; (f \; y) \; : \; ys &= \lambda y \; ys \; \rightarrow \; (:) \; (f \; y) \; ys && \text{haciendo prefijo a :} \\
&= \lambda y \; \rightarrow \; (:) \; (f \; y) && \text{eta reducción} \\
&= \lambda y \; \rightarrow \; ((:) \; . \; f) \; y && \text{def. de composición} \\
&= ((:) \; . \; f) && \text{eta reducción}
\end{aligned}$$

Por lo que, la función *map* puede reescribirse como

$$\text{map f} = \text{foldr ((:) . f) []}$$

Así,

$$\text{map length} = \text{foldr ((:) . length) []}$$

Regresando al lado derecho original, hay que tener en cuenta que

$$\text{product} = \text{foldr } (*) \text{ 1}$$

Sabiendo esto, la expresión a la que queremos llegar se vería de la forma

$$\text{foldr } (*) \text{ 1 . foldr ((:) . length) []} = \text{foldr h b}$$

Por el teorema de fusión de foldr, basta que se cumplan las siguientes dos condiciones (sabemos que product es estricta):

```
            foldr (*) 1 []              = b
            foldr (*) 1 (length x : xs) = h x (foldr (*) 1 xs)

```

De la primera ecuación obtenemos que $b = 1$ por la definición de *foldr*. De la segunda obtenemos

$$\begin{gathered}
\text{foldr } (*) \text{ 1 (length x : xs)} = \text{h x (foldr } (*) \text{ 1 xs)} \\
\Leftrightarrow \text{def. de foldr} \\
(*) \text{ (length x) (foldr } (*) \text{ 1 xs)} = \text{h x (foldr } (*) \text{ 1 xs)} \\
\Leftrightarrow \text{generalizando (foldr } (*) \text{ 1 xs) como } ys \\
(*) \text{ (length x) ys} = \text{h x ys} \\
\Leftrightarrow \text{por extensionalidad de funciones} \\
(*) \text{ (length x)} = \text{h x} \\
\Leftrightarrow \text{def. de composición de funciones} \\
((*) \text{ . length) x} = \text{h x} \\
\Leftrightarrow \text{por extensionalidad de funciones} \\
((*) \text{ . length)} = \text{h}
\end{gathered}$$

Por lo tanto,

```
            product . map length = foldr ((*) . length) 1

```

Finalmente, podemos concluir que la ecuación original es cierta ya que ambos lados de la igualdad son idénticos.