Facultad de Ciencias, UNAM
Programación Declarativa
Tarea 2: The Imperative is Dark and Full of Terrors

Rubí Rojas Tania Michelle

24 de febrero de 2020

1. Demuestra las siguientes propiedades

   - sum . map double = double . sum

     Demostración. Inducción estructural sobre xs.

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

$$
\begin{aligned}
(sum\ .\ map\ double)\ [] &= sum\ (map\ double\ []) && \\
&= sum\ [] && \text{def. de map} \\
&= 0 && \text{def. de sum} \\
&= 2 * 0 && \text{aritmética} \\
&= double\ 0 && \text{def. de double} \\
&= double\ (sum\ []) && \text{def. de sum} \\
&= (double\ .\ sum)\ [] &&
\end{aligned}
$$

     - Hipótesis de inducción.

       sum . map double = double . sum

     - Paso inductivo.

$$
\begin{aligned}
(sum\ .\ map\ double)\ (x : xs) &= sum\ (map\ double\ (x : xs)) && \\
&= sum\ (double\ x\ :\ map\ double\ xs) && \text{def. de map} \\
&= double\ x + sum\ (map\ double\ xs) && \text{def. de sum} \\
&= double\ x + double\ (sum\ xs) && \text{hipótesis de inducción} \\
&= 2 * x + 2 * (sum\ xs) && \text{def. de double} \\
&= 2 * (x + sum\ xs) && \text{distributividad} \\
&= double\ (x + sum\ xs) && \text{def. de double} \\
&= double\ (sum\ (x : xs)) && \text{def. de sum} \\
&= (double\ .\ sum)\ (x : xs) &&
\end{aligned}
$$

$\square$

- sum . map sum = sum . concat

Demostración. Inducción estructural sobre xs.

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

$$\begin{aligned}(sum\ .\ map\ sum)\ [] &= sum\ (map\ sum\ []) \\ &= sum\ [] && \text{def. de map} \\ &= sum\ (concat\ []) && \text{def. de concat} \\ &= (sum\ .\ concat)\ []\end{aligned}$$

- Hipótesis de inducción.

$$\text{sum . map sum = sum . concat}$$

- Paso inductivo.

$$\begin{aligned}(sum\ .\ map\ sum)\ (x : xs) &= sum\ (map\ sum\ (x : xs)) \\ &= sum\ (sum\ x\ :\ map\ sum\ xs) \\ &= sum\ x + sum\ (map\ sum\ xs) && \text{def. de sum} \\ &= sum\ x + sum\ (concat\ xs) && \text{hipótesis de inducción} \\ &= sum\ (x\ ++\ concat\ xs) && \text{propiedad de sum} \\ &= sum\ (concat(x : xs)) && \text{def. de concat} \\ &= (sum\ .\ concat)\ (x : xs)\end{aligned}$$

$\square$

- sum . sort = sum

Demostración. Inducción fuerte sobre la longitud de la lista.

- Caso base.
  $length\ xs = 0$. Esto quiere decir que la lista es vacía, por lo que este caso se cumple pues

$$\begin{aligned}(sum\ .\ sort)\ [] &= sum\ (sort\ []) \\ &= sum\ [] && \text{def. de sort}\end{aligned}$$

- Hipótesis de inducción.
  Supongamos que se cumple para todas las listas de longitud menor a (x:xs).
- Paso inductivo. Notemos que la conmutatatividad de la suma hace que esto funcione. Sea sort = quickSort, entonces

$$\begin{aligned}(sum\ .\ sort)\ (x : xs) &= sum\ (sort\ (x : xs)) \\ &= sum\ (sort\ (menorPrivote)\ ++\ [x] \\ &\quad ++\ sort\ (mayorIgualPivote)) && \text{def. de quickSort} \\ &= sum\ (sort\ (menorPrivote))\ +\ sum\ [x] \\ &\quad ++\ sum(sort(mayorIgualPivote)) && \textit{prop. de sum} \\ &= sum\ (menorPrivote)\ +\ sum\ [x] + \\ &\quad sum(mayorIgualPivote) && \text{H.I.} \\ &= sum(menorPrivote\ ++\ [x]\ ++\ mayorIgualPivote) && \text{prop. de sum} \\ &= sum(x\ : menorPrivote\ ++\ mayorIgualPivote) && \text{prop. de ++} \\ &= sum(x : xs) && \text{def. de ++}\end{aligned}$$

$\square$

2. Demuestra o da un contraejemplo de cada una de las siguientes propiedades.

- take n xs ++ drop n xs = xs

  Demostración. Inducción sobre n.

  - Caso base.
    $n = 0$. Este caso se cumple ya que

    $$\begin{aligned} take\ 0\ xs\ ++\ drop\ 0\ xs &= []\ ++\ xs && \text{def. de take y drop} \\ &= xs && \text{def. de ++} \end{aligned}$$

  - Hipótesis de inducción.

    take n xs ++ drop n xs = xs

  - Paso inductivo.
    Tenemos dos casos

    i) $n = n + 1 \wedge xs = []$

    $$\begin{aligned} take\ (n+1)\ []\ ++\ drop\ (n+1)\ [] &= []\ ++\ [] && \text{def. de take y drop} \\ &= [] && \text{def. de ++} \end{aligned}$$

    ii) $n = n + 1 \wedge xs = (x : xs)$

    $$\begin{aligned} take\ (n+1)\ (x:xs)\ ++\ drop\ (n+1)\ (x:xs) &= (x\ :\ take\ n\ xs)\ ++\ (drop\ n\ xs) \\ &= x\ :\ ((take\ n\ xs)\ ++\ (drop\ n\ xs)) \\ &= (x:xs) \end{aligned}$$

  □

- take m . take n = take (min m n)

  Demostración. Sin pérdida de generalidad, asumimos que $m, n \geq 0$.
  Inducción sobre m.

  - Caso base.
    $m = 0$. Este caso se cumple ya que

    $$\begin{aligned} (take\ 0\ .\ take\ n)\ xs &= take\ 0\ (take\ n\ xs) \\ &= [] && \text{def. de take} \\ &= take\ 0\ xs && \text{def. de take} \\ &= take\ (min\ 0\ n)\ xs && \text{def. de min} \\ &= (take\ (min\ 0\ n))\ xs \end{aligned}$$

  - Hipótesis de inducción.

    take m . take n = take (min m n)

  - Paso inductivo.
    Tenemos dos casos

    i) $m = m + 1 \wedge xs = []$

    $$\begin{aligned} (take\ (m+1)\ .\ take\ n)\ [] &= take\ (m+1)\ (take\ n\ []) \\ &= take\ (m+1)\ [] && \text{def. de take} \\ &= [] && \text{def. de take} \\ &= take\ (min\ (m+1)\ n)\ [] && \text{def. de take} \\ &= (take\ (min\ (m+1)\ n))\ [] \end{aligned}$$

ii) $n = n+1 \wedge xs = (x : xs)$
Recordemos que *min (x y) = min ((x+1) (y+1))*, es decir, el mínimo seguirá siendo el mínimo si es que a los números x, y se les suma una unidad.

$$\begin{aligned}
(take\ (m+1)\ .\ take\ n)\ (x : xs) &= take\ (m+1)\ (take\ n\ (x : xs)) \\
&= take\ (m+1)\ (x\ :\ take\ (n-1)\ xs) && \text{def. de take} \\
&= x\ :\ take\ m\ (take\ (n-1)\ xs) && \text{def. de take} \\
&= x\ :\ take\ (min\ m\ (n-1))\ xs && \text{H. I.} \\
&= take\ (min\ (m+1)\ n)\ (x : xs) && \text{def. de take} \\
&= (take\ (min\ (m+1)\ n))\ (x : xs)
\end{aligned}$$

□

- map f . take n = take n . map f

Demostración. Inducción sobre n.

- Caso base.
  $n = 0$. Este caso se cumple ya que

$$\begin{aligned}
(map\ .\ take\ 0)\ xs &= map\ f\ (take\ 0\ xs) \\
&= map\ f\ [] && \text{def. de take} \\
&= [] && \text{def. de map} \\
&= take\ 0\ (map\ f\ xs) && \text{def. de take} \\
&= (take\ 0\ .\ map\ f)\ xs
\end{aligned}$$

- Hipótesis de inducción.

map f . take n = take n . map f

- Paso inductivo. Tenemos dos casos

i) $n = n+1 \wedge xs = []$

$$\begin{aligned}
(map\ .\ take\ (n+1))\ [] &= map\ f\ (take\ (n+1)\ []) \\
&= map\ f\ [] && \text{def. de take} \\
&= [] && \text{def. de map} \\
&= take\ (n+1)\ [] && \text{def. de take} \\
&= take\ (n+1)\ (map\ f\ []) && \text{def. de map} \\
&= (take\ (n+1)\ .\ map\ f)\ []
\end{aligned}$$

ii) $n = n+1 \wedge xs = (x : xs)$

$$\begin{aligned}
(map\ .\ take\ (n+1))\ (x : xs) &= map\ f\ (take\ (n+1)\ (x : xs)) \\
&= map\ f\ (x\ :\ take\ n\ xs) && \text{def. de take} \\
&= f\ x\ :\ map\ f\ (take\ n\ xs) && \text{def. de map} \\
&= f\ x\ :\ take\ n\ (map\ f\ xs) && \text{hipótesis de inducción} \\
&= take\ (n+1)\ (f\ x\ :\ map\ f\ xs) && \text{def. de take} \\
&= take\ (n+1)\ (map\ f\ (x : xs)) && \text{def. de map} \\
&= (take\ (n+1)\ .\ map\ f)\ (x : xs)
\end{aligned}$$

□

- filter p . concat = concat . map (filter p)

  Demostración. Inducción sobre xs.

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

$$
\begin{aligned}
(filter\ p\ .\ concat)\ [] &= filter\ p\ (concat\ []) & \\
&= filter\ p\ [] & \text{def. de concat} \\
&= [] & \text{def. de filter} \\
&= concat\ [] & \text{def. de concat} \\
&= concat\ (map\ (filter\ p)\ []) & \text{def. de map} \\
&= (concat\ .\ map\ (filter\ p))\ [] &
\end{aligned}
$$

  - Hipótesis de inducción.

    filter p . concat = concat . map (filter p)

  - Paso inductivo.

$$
\begin{aligned}
(filter\ p\ .\ concat)\ (x : xs) &= filter\ p\ (concat\ (x : xs)) & \\
&= filter\ p\ (x\ ++\ concat\ xs) & \text{def. de concat} \\
&= (filter\ p)\ x\ ++\ filter\ p\ (concat\ xs) & \text{distributividad} \\
&= (filter\ p)\ x\ ++\ concat\ (map\ (filter\ p)\ xs) & \text{hipótesis de inducción} \\
&= concat\ ((filter\ p)\ x\ :\ map\ (filter\ p)\ xs) & \text{def. de concat} \\
&= concat\ (map\ (filter\ p)\ (x : xs)) & \text{def. de map} \\
&= (concat\ .\ map\ (filter\ p))\ (x : xs) &
\end{aligned}
$$

  □

3. Consideremos la siguiente afirmación

   map (f . g) xs = map f $ map g xs

   (a) ¿Se cumple para cualquier xs? Si es cierta bosqueja la demostración, en caso contrario, ¿qué condiciones se deben pedir sobre xs para que sea cierta?

   SOLUCIÓN: Sí, se cumple para cualquier lista xs. Para justificarlo, demostraremos la propiedad usando inducción estructural.

   Demostración. Inducción estructural sobre xs.

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

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

   - Hipótesis de inducción.

     map (f . g) xs = map f $ map g xs

- Paso inductivo.

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

□

(b) Intuitivamente, ¿qué lado de la desigualdad resulta más eficiente? ¿Esto es cierto incluso en lenguajes con evaluación perezosa? Justifica ambas respuestas.

SOLUCIÓN: La expresión **map f $ map g xs** hace dos recorridos a la lista xs al momento de ejecutarse (un recorrido por cada función **map**), mientras que **map (f . g) xs** hace un sólo recorrido de la lista xs, el cual es realizado por nuestra única función **map**. Es decir, **map (f . g) xs** hace la *optimización* de reemplazar dos recorridos de la lista xs por uno solo.

Esto es cierto incluso para lenguajes con evaluación perezosa ya que en ambos casos debemos evaluar todos los valores de la lista, pues es necesario gracias a la aplicación de la función **map**. Así que si podemos evitar recorrer la lista más de una vez, es mejor para nosotros (y nuestra complejidad en espacio).