

Programación declarativa. Tarea 2

The Imperative is Dark and Full of Terrors

Juan Alfonso Garduño Solís
Emiliano Galeana Araujo

Facultad de ciencias, UNAM

Fecha de entrega: Lunes 24 de febrero de 2020

1 Demostraciones de propiedades

```
1 sum . map double = double . sum
2
```

– Caso base:

```
1 sum . map double [] = double . sum []
2 sum . 0 = double . sum []
3 0 = double . sum []
4 sum [] = double sum []
5 double . sum [] = double . sum []
6
```

– Hipótesis

```
1 sum . map double xs = double . sum xs
2
```

– Paso inductivo

– Por demostrar

```
1 sum . map double (x:xs) = double . sum (x:xs)
2
```

```
1 sum . map double (x:xs) = double . sum (x:xs)
2 -- Definicion de aplicar double a la cabeza de x:xs.
3 sum . [double x] ++ map double xs = double . sum (x:xs)
4 -- Definicion de aplicar sum ([double x] es igual a [2*x]).
5 2*x + sum . map double xs = double . sum (x:xs)
6 -- Hipotesis.
7 2*x + double . sum xs = double . sum (x:xs)
8 -- Defincion de double inversa.
9 double x + sum xs = double . sum (x:xs)
10 -- Metemos x a la funcion sum.
11 double . sum (x:xs) = double . sum (x:xs)
12
```

```

1 sum . map sum = sum . concat
2

```

– Caso base:

```

1 sum . map sum [] = sum . concat []
2 sum [] = sum . concat []
3 0 = sum . concat []
4 sum [] = sum . concat []
5 sum . concat [] = sum . concat []
6

```

– Hipótesis

```

1 sum . map sum xs = sum . concat xs
2

```

– Paso inductivo

– Por demostrar

```

1 sum . map sum (x:xs) = sum . concat (x:xs)
2

```

```

1 sum . map sum (x:xs) = sum . concat (x:xs)
2 -- Definicion de aplicar map sum a la cabeza de x:xs. Definimos
  sum' x como
3 el resultado de sum x.
4 sum [sum' x] . map sum xs = sum . concat (x:xs)
5 -- Definicion de aplicar sum a una lista con un elemento (sum'
  x).
6 (sum' x) + sum . map xs = sum . concat (x:xs)
7 -- Hipotesis.
8 (sum' x) + sum . concat (xs) = sum . concat (x:xs)
9 -- Metemos la suma a la funcion sum.
10 sum [sum' x] . concat (xs) = sum . concat (x:xs)
11 -- Como [sum' x] es el resultado de la aplicar sum a la lista
  x. Siendo x
12 una lista, podemos hacer lo siguiente.
13 sum . concat (x:xs) = sum . concat (x:xs)
14

```

```

1 sum . sort = sum
2

```

– Caso base:

```

1 sum . sort [] = sum []
2 sum [] = sum []
3

```

– Hipótesis

```

1 sum . sort xs = sum xs
2

```

– Paso inductivo

– Por demostrar

```

1      sum . sort (x:xs) = sum (x:xs)
2      -- Ordenar la cola de una lista y luego ordenar la cola
   (ordenada)
3      concatenada a la cabeza es lo mismo que ordenar toda la
   lista.
4      sum [x] ++ sort xs = sum (x:xs)
5      -- Definicion de sum.
6      x + sum . sort xs = sum (x:xs)
7      -- Hipotesis.
8      x + sum xs = sum (x:xs)
9      -- Definicion inversa de sum.
10     sum [x]++xs = sum (x:xs)
11     -- Definicion de concatenar.
12     sum (x:xs) = sum (x:xs)
13

```

Donde, double se define de la siguiente manera:

```

1  double :: Integer -> Integer
2  double x = 2 * x

```

Y, sum,map,sort y concat son las definidas en el Prelude, de Haskell.

2 Función take

En Haskell la función `take n` toma los primeros `n` elementos de una lista, mientras que `drop n` regresa la lista sin los primeros `n` elementos de esta. Demuesrta o da un contraejemplo:

```

1  take n xs ++ drop n xs = xs
2

```

```

1  take m . take n = take (min m n)
2

```

```

1  map f . take n = take n . map f
2

```

```

1  filter p . concat = concat . map (filter p)
2

```

3 Función map

Consideremos la siguiente afirmación

```

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

```

- ¿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?
- Intuitivamente, ¿Qué lado de la igualdad resulta más eficiente? ¿Esto es cierto incluso en lenguajes con evaluación perezosa? Justifica tu respuesta.