Introducción a la Programación

Guía Práctica 3
Introducción a Haskell

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

1. Definición de funciones básicas

Ejercicio 1. a) Implentar la función parcial `f :: Integer ->Integer` definida por extensión de la siguiente manera:

$$f(1) = 8$$
$$f(4) = 131$$
$$f(16) = 16$$

cuya especificación es la siguiente:

`problema` `f` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
`requiere`: $\{n = 1 \lor n = 4 \lor n = 16\}$
`asegura`: $\{(n = 1 \rightarrow result = 8) \land (n = 4 \rightarrow result = 131) \land (n = 16 \rightarrow result = 16)\}$
}

b) Análogamente, especificar e implementar la función parcial `g :: Integer ->Integer`

$$g(8) = 16$$
$$g(16) = 4$$
$$g(131) = 1$$

c) A partir de las funciones definidas en los ítems 1 y 2, implementar las funciones parciales $h = f \circ g$ y $k = g \circ f$

Ejercicio 2. ★ Especificar e implementar las siguientes funciones, incluyendo su signatura.

a) `absoluto`: calcula el valor absoluto de un número entero.

b) `maximoabsoluto`: devuelve el máximo entre el valor absoluto de dos números enteros.

c) `maximo3`: devuelve el máximo entre tres números enteros.

d) `algunoEs0`: dados dos números racionales, decide si alguno de los dos es igual a 0 (hacerlo dos veces, una usando *pattern matching* y otra no).

e) `ambosSon0`: dados dos números racionales, decide si ambos son iguales a 0 (hacerlo dos veces, una usando *pattern matching* y otra no).

f) `mismoIntervalo`: dados dos números reales, indica si están relacionados considerando la relación de equivalencia en $\mathbb{R}$ cuyas clases de equivalencia son: $(-\infty, 3]$, $(3, 7]$ y $(7, \infty)$, o dicho de otra forma, si pertenecen al mismo intervalo.

g) `sumaDistintos`: que dados tres números enteros calcule la suma sin sumar repetidos (si los hubiera).

h) `esMultiploDe`: dados dos números naturales, decidir si el primero es múltiplo del segundo.

i) `digitoUnidades`: dado un número natural, extrae su dígito de las unidades.

j) `digitoDecenas`: dado un número natural, extrae su dígito de las decenas.

Ejercicio 3. Implementar una función `estanRelacionados :: Integer ->Integer ->Bool`

`problema` `estanRelacionados` (a:$\mathbb{Z}$, b:$\mathbb{Z}$) : Bool {
`requiere`: $\{a \neq 0 \land b \neq 0\}$
`asegura`: $\{(res = true) \leftrightarrow a * a + a * b * k = 0$ para algún $k \in \mathbb{Z}$ con $k \neq 0)\}$
}

Por ejemplo:

`estanRelacionados 8 2` $\rightsquigarrow$ `True` porque existe un $k = -4$ tal que $8^2 + 8 \times 2 \times (-4) = 0$.
`estanRelacionados 7 3` $\rightsquigarrow$ `False` porque no existe un k entero tal que $7^2 + 7 \times 3 \times k = 0$.

Ejercicio 4. ★
Especificar e implementar las siguientes funciones utilizando tuplas para representar pares, ternas de números.

a) `prodInt`: calcula el producto interno entre dos tuplas $\mathbb{R} \times \mathbb{R}$.

b) `todoMenor`: dadas dos tuplas $\mathbb{R} \times \mathbb{R}$, decide si es cierto que cada coordenada de la primera tupla es menor a la coordenada correspondiente de la segunda tupla.

c) `distanciaPuntos`: calcula la distancia entre dos puntos de $\mathbb{R}^2$.

d) `sumaTerna`: dada una terna de enteros, calcula la suma de sus tres elementos.

e) `sumarSoloMultiplos`: dada una terna de números enteros y un natural, calcula la suma de los elementos de la terna que son múltiplos del número natural. *Por ejemplo:*
`sumarSoloMultiplos (10,-8,-5) 2` $\rightsquigarrow$ `2`
`sumarSoloMultiplos (66,21,4) 5` $\rightsquigarrow$ `0`
`sumarSoloMultiplos (-30,2,12) 3` $\rightsquigarrow$ `-18`

f) `posPrimerPar`: dada una terna de enteros, devuelve la posición del primer número par si es que hay alguno, y devuelve 4 si son todos impares.

g) `crearPar :: a ->b ->(a, b)`: crea un par a partir de sus dos componentes dadas por separado (debe funcionar para elementos de cualquier tipo).

h) `invertir :: (a, b) ->(b, a)`: invierte los elementos del par pasado como parámetro (debe funcionar para elementos de cualquier tipo).

Ejercicio 5. Implementar la función `todosMenores :: (Integer, Integer, Integer) ->Bool`

`problema todosMenores` $((n_1,n_2,n_3) : \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z})$: Bool {
  `requiere:` $\{True\}$
  `asegura:` $\{(res = true) \leftrightarrow ((f(n_1) > g(n_1)) \wedge (f(n_2) > g(n_2)) \wedge (f(n_3) > g(n_3))))\}$
}

`problema f` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` $\{True\}$
  `asegura:` $\{(n \leq 7 \rightarrow res = n^2) \wedge (n > 7 \rightarrow res = 2n - 1)\}$
}

`problema g` (n: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere:` $\{True\}$
  `asegura:` {Si n es un número par, entonces $res = n/2$, en caso contrario, $res = 3n + 1$}
}

Ejercicio 6. Programar una función `bisiesto :: Integer ->Bool` según la siguiente especificación:

`problema bisiesto` (año: $\mathbb{Z}$) : Bool {
  `requiere:` $\{True\}$
  `asegura:` {$res = false \leftrightarrow$ año no es múltiplo de 4 o año es múltiplo de 100 pero no de 400}
}

Por ejemplo:
`bisiesto 1901` $\rightsquigarrow$ `False`,   `bisiesto 1904` $\rightsquigarrow$ `True`,
`bisiesto 1900` $\rightsquigarrow$ `False`,   `bisiesto 2000` $\rightsquigarrow$ `True`.

Ejercicio 7. Implementar una función:
`distanciaManhattan:: (Float, Float, Float) ->(Float, Float, Float) ->Float`

`problema distanciaManhattan` $(p : \mathbb{R} \times \mathbb{R} \times \mathbb{R}, q : \mathbb{R} \times \mathbb{R} \times \mathbb{R}) : \mathbb{R}$ {
  `requiere:` $\{True\}$
  `asegura:` $\{res = \sum_{i=0}^{2} |p_i - q_i|\}$
}

Por ejemplo:
`distanciaManhattan (2, 3, 4) (7, 3, 8)` $\rightsquigarrow$ `9`
`distanciaManhattan ((-1), 0, (-8.5)) (3.3, 4, (-4))` $\rightsquigarrow$ `12.8`

Ejercicio 8. Implementar una función `comparar :: Integer ->Integer ->Integer`

`problema` `comparar` (a:$\mathbb{Z}$, b:$\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere`: $\{True\}$
  `asegura`: $\{(res = 1 \leftrightarrow sumaUltimosDosDigitos(a) < sumaUltimosDosDigitos(b))\}$
  `asegura`: $\{(res = -1 \leftrightarrow sumaUltimosDosDigitos(a) > sumaUltimosDosDigitos(b))\}$
  `asegura`: $\{(res = 0 \leftrightarrow sumaUltimosDosDigitos(a) = sumaUltimosDosDigitos(b)))\}$
}

`problema` `sumaUltimosDosDigitos` (x: $\mathbb{Z}$) : $\mathbb{Z}$ {
  `requiere`: $\{True\}$
  `asegura`: $\{res = (x \mod 10) + (\lfloor(x/10)\rfloor \mod 10)\}$
}

Por ejemplo:

`comparar 45 312` $\rightsquigarrow$ `-1` porque $312 \prec 45$ y $1+2 < 4+5$.
`comparar 2312 7` $\rightsquigarrow$ `1` porque $2312 \prec 7$ y $1+2 < 0+7$.
`comparar 45 172` $\rightsquigarrow$ `0` porque no vale $45 \prec 172$ ni tampoco $172 \prec 45$.

Ejercicio 9. A partir de las siguientes implementaciones en Haskell, describir en lenguaje natural qué hacen y especificarlas semiformalmente.

a)
```
f1 :: Float -> Float
f1 n | n == 0 = 1
     | otherwise = 0
```

b)
```
f2 :: Float -> Float
f2 n | n == 1  =  15
     | n == -1 = -15
```

c)
```
f3 :: Float -> Float
f3 n | n <= 9 = 7
     | n >= 3 = 5
```

d)
```
f4 :: Float -> Float -> Float
f4 x y = (x+y)/2
```

e)
```
f5 :: (Float, Float) -> Float
f5 (x, y) = (x+y)/2
```

f)
```
f6 :: Float -> Int -> Bool
f6 a b = truncate a == b
```