Programación para la Inteligencia Artificial

Haskell - Práctica 3

Tipos y clases de tipos

Definición de clases de números

A. Racionales

El tipo *Racional* se puede representar con dos enteros, de forma que uno de ellos represente al numerador y el otro al denominador. Pueden definirse las siguientes operaciones:

- *simplificaRac*: Obtiene la representación canónica de un racional. Por ejemplo:

```
simplificaRac 6/12 = 1/2
simplificaRac 6/-12 = -1/2
simplificaRac -6/-12 = 1/2
simplificaRac -6/12 = -1/2
```

- *multRac*: Multiplica dos racionales. Por ejemplo:

```
multRac 1/4 4/5 = 1/5
```

- *divRac*: Divide dos racionales. Por ejemplo:

```
divRac 1/4 4/5 = 5/16
```

- *sumRac*: Suma dos racionales. Por ejemplo:

```
sumRac 1/4 4/5 = 21/20
```

- *resRac*: Resta dos racionales. Por ejemplo:

```
resRac 1/4 4/5 = -11/20
```

Define en Haskell el tipo *Racional* de dos formas diferentes:

1. Como renombramiento de un par de enteros `(Int,Int)`. Definir las funciones anteriores[1]. Definir, además, una función *muestraRac* que convierta un *Racional* en su cadena correspondiente. Por ejemplo:

   ```
   muestraRac (6,12) = "1/2"
   muestraRac (6,6) = "1"
   muestraRac (multRac (1,2)(2,3)) = "1/3"
   ```

2. Como un nuevo tipo de datos, por ejemplo:

   ```
   data Racional = Rac Integer Integer
   ```

   Definir las funciones anteriores[1]. Definir, además, *Racional* como una instancia de `Show`, de forma que la función `show` muestre la forma simplificada obtenida mediante la función `simplificaRac` definida al principio.

3. Completar el ejercicio anterior declarando el tipo *Racional* como una instancia de la clase `Num`, redefiniendo las operaciones `(*)`, `(+)`, `(-)`, `negate` (que modifica el signo a un número racional), `fromInteger`[2] (que convierte un entero en racional), `signum` y `abs`.

B. Naturales

Supongamos que tenemos la siguiente definición:

```
data Nat = Cero | Succ Nat
      deriving (Eq,Show)
```

1. Declara el tipo `Nat` como una instancia de la clase `Num`, redefiniendo las operaciones `(+)`, `(-)`,`(*)`, `abs`, `signum` y `fromInteger`.

2. Declara el tipo `Nat` como una instancia de la clase `Ord`, redefiniendo la operación `(<)`.

3. Define la función

   ```
   divModN ::  Nat -> Nat -> (Nat,Nat)
   ```

   que devuelva el cociente y el módulo de dos naturales.

4. Declara el tipo `Nat` como instancia de la clase `Show`, de manera que se visualicen los naturales con las cadenas "0", "1", "2",...

[1] Pueden usarse las funciones predefinidas `signum`, que devuelve `-1` si el número es negativo y `1` en otro caso, `abs`, que devuelve el valor absoluto de un número, `gcd`, que devuelve el *máximo común divisor* de dos números y `lcm` que devuelve el *mínimo común múltiplo*.

[2] Ojo al evaluar esta función, puesto que ya está predefinida. Para asegurarnos de que se evalúa la nuestra, al usarla podemos indicar el tipo explícitamente, por ejemplo:
`fromInteger 7::Racional`