Práctica 8
Programación Funcional - Tipos Compuestos

Algoritmos y Estructura de Datos I

Segundo Cuatrimestre 2010

1 Tipos Compuestos

En los ejercicios de tipos compuestos NO TIENEN que definir los tipos en Haskell. Pueden hacerlo si gustan, pero no es necesario que lo hagan. Solamente tienen que programar las funciones que se explicitan.

Ejercicio 1. Recordemos que el tipo *Vector* se define como

tipo Vector{
  observador abscisa($t : Vector$) : $\mathbb{R}$
  observador ordenada($t : Vector$) : $\mathbb{R}$
}

Las funciones en Haskell que implementan los observadores del tipo son:

1. *abscisa* :: *Vector* → *Float*
2. *ordenada* :: *Vector* → *Float*

Se pide implementar los siguientes problemas:

1. problema igualX(v_1, v_2 : Vector) = *result* : Bool.
2. problema igualY(v_1, v_2 : Vector) = *result* : Bool.
3. problema colineales(v_1, v_2 : Vector) = *result* : Bool.
4. problema módulo(v : Vector) = *result* : $\mathbb{R}$.
5. problema productoEscalar(v_1, v_2 : Vector) = *result* : $\mathbb{R}$.

Ejercicio 2. Se tiene definido el tipo *MatrizCuadrada* (de enteros no negativos)como:

tipo MatrizCuadrada{
  observador dimension($m : MatrizCuadrada$) : $\mathbb{Z}$
  observador valor($t : MatrizCuadrada, f : Fila, c : Columna$) : $\mathbb{Z}$\{
    requiere : $1 \leq f \leq$ dimension(m) $\wedge\ 1 \leq c \leq$ dimension(m);
  }

}

tipo `Fila = Int`
tipo `Columna = Int`

Las funciones de Haskell que implementan los observadores del tipo son:

- *dimensión* :: *MatrizCuadrada* → *Integer*, que devuelve el tamaño de la matriz.
- *valor* :: *MatrizCuadrada* → *Fila* → *Columna* → *Integer*, que devuelve el valor de la casilla indicada. Esta función está indefinida cuando *Fila* y *Columna* están fuera del rango válido. Los tipos *Fila* y *Columna* son sinónimos de *Integer*.

Por ejemplo, si m es la matriz cuadrada

$$\begin{bmatrix} 3 & 1 & 6 \\ 2 & 5 & 3 \\ 5 & 4 & 1 \end{bmatrix}$$

entonces *dimension* m es 3, *valor* m 1 2 es 1 y *valor* m 3 2 es 4.

1. Especificar y definir las siguientes operaciones:
   (a) *todosIguales* :: *MatrizCuadrada* → *Bool*, que indica si todos los elementos de la matriz cuadrada tienen el mismo valor.
   (b) *identidad* :: *MatrizCuadrada* → *Bool*, que indica si la matriz es una matriz identidad. La matriz identidad tiene 0 en todas las posiciones, excepto en las posiciones de la diagonal, que tienen 1.
   (c) *máximo* :: *MatrizCuadrada* → *Integer*, que devuelve el valor máximo de la matriz.
   (d) *mínimo* :: *MatrizCuadrada* → *Integer*, que devuelve el valor mínimo de la matriz.
2. Redefinir la función del item a, ahora usando solamente los items c y d.
3. Una matriz cuadrada es *degradé* si tiene las siguientes propiedades :
   - Todas las celdas que pertenecen a una misma diagonal de la matriz contiene el mismo número natural.
   - El número que contiene una diagonal es mayor que los números que contienen todas las diagonales que están más a la izquierda. Es decir, las diagonales están ordenadas descendentemente.

   Un ejemplo de matriz cuadrada *degradé* es:

$$\begin{bmatrix} 4 & 6 & 25 & 75 & 99 \\ 3 & 4 & 6 & 25 & 75 \\ 2 & 3 & 4 & 6 & 25 \\ 1 & 2 & 3 & 4 & 6 \\ 0 & 1 & 2 & 3 & 4 \end{bmatrix}$$

   Las siguientes matrices no son *degradé*:

$$\begin{bmatrix} 4 & 6 & 25 & 75 \\ 3 & 4 & 6 & 25 \\ 8 & 3 & 4 & 6 \\ 1 & 8 & 3 & 4 \end{bmatrix} \quad \begin{bmatrix} 99 & 8 & 7 \\ 8 & 7 & 3 \\ 7 & 3 & 1 \end{bmatrix} \quad \begin{bmatrix} 7 & 11 & 13 \\ 2 & 8 & 11 \\ 1 & 2 & 9 \end{bmatrix}$$

   Se pide definir la función *esDegradé* que dada una matriz cuadrada devuelva *True* si es *degradé* y *False* en caso contrario.

Ejercicio 3. Se cuenta con el tipo compuesto *Polinomio* (de coeficientes enteros), definido de la siguiente manera:

```
tipo Polinomio{
   observador grado(Polinomio) : ℤ
   observador coeficiente(P : Polinomio, n : ℤ) : ℤ{
        requiere : n ≥ 0;
   }
   invariante grado(p) ≥ 0
}
```

Las funciones de Haskell que implementan los observadores del tipo son:

- *grado* :: *Polinomio* → *Integer*
- *coeficiente* :: *Polinomio* → *Integer* → *Integer*

Por ejemplo:

$$\begin{aligned} \textit{grado}\ (2x^3+8) &= 3 \\ \textit{coeficiente}\ (2x^3+8)\ 0 &= 8 \\ \textit{coeficiente}\ (2x^3+1)\ 1 &= 0 \\ \textit{coeficiente}\ (2x^3+1)\ 3 &= 2 \end{aligned}$$

Especificar y definir la función *evaluar* :: *Polinomio* → *Integer* → *Integer*, que devuelve el valor del polinomio en el punto dado.

Ejercicio 4. Se define el tipo compuesto *Pirámide* como:

tipo Pirámide{
  observador altura(Pirámide) : $\mathbb{Z}$
  observador valor(P:Pirámide, $f : \mathbb{Z}, c : \mathbb{Z}$) : $\mathbb{Z}${
    requiere : $1 \leq c \leq f \leq$ altura(P);
  }

}

Una pirámide está bien formada cuando cada celda es igual a la suma de las dos celdas inferiores.
Las funciones en Haskell que implementan los observadores del tipo son:

1. *altura* :: *Piramide* → *Int*
2. *valor* :: *Piramide* → *Int* → *Int* → *Int*

Se pide definir la función *bienFormada* :: *Pirámide* → *Bool*, que indica si una pirámide está bien formada.

Ejercicio 5. Se define el tipo compuesto *Tablero* como:

tipo Tablero{
  observador cantFilas($Tablero$) : $\mathbb{Z}$
  observador cantColumnas($Tablero$) : $\mathbb{Z}$
  observador colorCasilla($T : Tablero, f : \mathbb{Z}, c : \mathbb{Z}$) : $Color${
    requiere : $1 \leq f \leq$ cantFilas(T) $\wedge\ 1 \leq c \leq$ cantColumnas(T);
  }

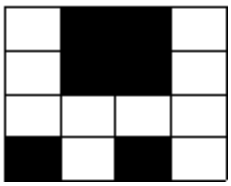
}

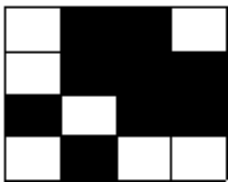
Los elementos del tipo enumerado Color son Blanco y Negro.
Las funciones en Haskell que implementan los observadores del tipo son:

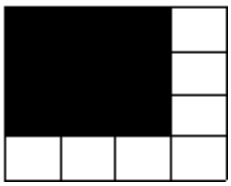
1. *cantFilas,cantColumnas* :: *Tablero* → *Int*
2. *colorCasilla* :: *Tablero* → *Int* → *Int* → *Color*

Suponiendo que se cuenta con el siguiente algoritmo,
pintarNegro :: *Tablero* → *Fila* → *Columna* → *Tablero*, que devuelve un tablero igual al original salvo por la casilla indicada por los parámetros Fila y Columna, que ahora es negra. Si no es una casilla válida, devuelve un tablero igual al original.
Especificar y definir las siguientes funciones:

1. *cantidadBloquesNegros* :: *Tablero* → *Integer*. Esta función devuelve la cantidad de *bloques* negros que hay en el tablero, donde un *bloque* es un cuadrado de dos por dos casillas, todas negras. No importa que los bloques estén superpuestos. Ejemplos:

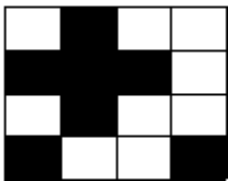

*Tab*1


*Tab*2

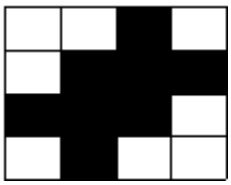

*Tab*3

$$
\begin{aligned}
cantidadBloquesNegros\ Tab1 &= 1 \\
cantidadBloquesNegros\ Tab2 &= 2 \\
cantidadBloquesNegros\ Tab3 &= 4
\end{aligned}
$$

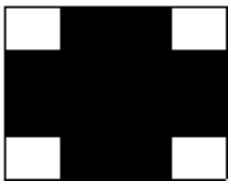
2. *cantidadCrucesNegras* :: *Tablero* → *Integer*. Esta función devuelve la cantidad de *cruces* negras que hay en el tablero, donde una *cruz* tiene tres casillas negras de largo por tres de ancho. No importa que las cruces estén superpuestas. Ejemplos:



*Tab*4



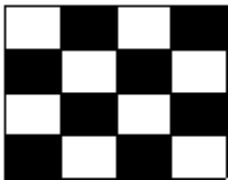
*Tab*5



*Tab*6

$$
\begin{aligned}
cantidadCrucesNegras\ Tab4 &= 1 \\
cantidadCrucesNegras\ Tab5 &= 2 \\
cantidadCrucesNegras\ Tab6 &= 4
\end{aligned}
$$

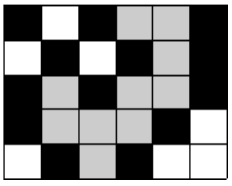
3. *esDamero* :: *Tablero* → *Bool*. Esta función verifica si el tablero alterna casillas blancas con negras en todas las posiciones (como un tablero de damas o de ajedrez). Ejemplo:



*Tab*7

$$esDamero\ Tab7 = True$$

4. (Opcional) *tamMayorSectorBlanco* :: *Tablero* → *Integer*. Esta función devuelve el tamaño del *sector blanco* más grande, donde un *sector blanco* es un conjunto de casillas blancas adyacentes, delimitadas por casillas negras o por el fin del tablero. Ejemplo:



*Tab*8

$tamMayorSectorBlanco\ Tab8 = 10$ (el sector blanco más grande es el que está sombreado)

Nota: Cuidado con las indefiniciones.

Ejercicio 6. Se cuenta con el tipo compuesto Documento que modela un documento de texto. Básicamente, sobre un documento de texto vacío, se pueden ir agregando palabras y puntos. Los puntos delimitan las oraciones. Suponer que todas las oraciones terminan en punto.

tipo Documento{
observador oraciones(d : *Documento*) : [[*String*]]
}

Se busca implementar en Haskell algunos problemas sobre el tipo compuesto Documento mediante el tipo algebraico `Documento`. Dicho tipo está definido con los siguientes constructores

```
data Documento = Blanco | Pal Documento String | Punto Documento
```

que permiten crear un documento vacío, agregar una palabra y un punto respectivamente. Por ejemplo, el documento "Hola mundo. Chau mundo." se contruye de la siguiente manera: `Punto (Pal (Pal (Punto (Pal (Pal Blanco "Hol`
Se cuenta también, con `reverso::String -> String` que devuelve el reverso de un String. (reverse "Parcial" devuelve "laicraP").

1. `oraciones :: Documento -> [[String]]`, que devuelve una lista de listas, correspondientes a cada una de las oraciones del documento. Nota: el Punto se representa con el String ".". Aplicar `oraciones` al documento "Hola mundo. Chau mundo." debería devolver `[ ["Hola","mundo","."],["Chau","mundo","."]]`

2. `textoSatanico :: Documento -> Documento`, que devuelve el documento original, pero escrito totalmente al reves. En el caso de que la última oración del texto satánico no termine con punto final, agregarlo.

   Por ejemplo: si el documento original era "Hola mundo. Chau mundo.", el textoSatanico correspondiente debería ser: ".odnum uahC. odnum aloH.". Observar que se agregó el punto a la última oración, para respetar que todas las oraciones terminen en punto.

Ejercicio 7. Se cuenta con el tipo compuesto Tateti que modela un tablero de Tateti. Se busca implementar en Haskell algunos problemas sobre este tipo compuesto mediante su tipo algebraico que se define con los siguientes constructores:

```
data Tateti = Nuevo | Cruz Int Tateti | Circulo Int Tateti
```

que permiten crear un Tablero nuevo (vacío), agregar una cruz o un círculo en la posición pasada como entero. Las posiciones del tablero se representan de la siguiente manera

```
1  2  3
4  5  6
7  8  9
```

Así, un tablero con una cruz en el centro se contruye de la siguiente manera: `Cruz 5 Nuevo`

1. `esValido :: Tateti -> Bool`, que devuelve verdadero sólo si se cumplen estas 3 condiciones a la vez:
   - todas las posiciones jugadas en el Tateti se encuentran entre 1 y 9,
   - las posiciones fueron ocupadas una sola vez -no existen dos jugadas en el Tateti que hayan utilizado la misma posición-,
   - las cruces y los círculos se jugaron alternadamente.

   Por ejemplo, aplicar `esValido` al Tateti `Cruz 1 (Cruz 5 Nuevo)` debería devolver Falso ya que Cruz jugó dos veces seguidas

2. `litoCruz :: Tateti -> [Int] ->Bool`, que toma un Tateti y una lista de posiciones válidas (no hay posiciones repetidas y sus valores están entre 1 y 9) y devuelve verdadero sólo si se cumplen a la vez que:
   - se trata de un tablero de Tateti válido (Hint: puede utilizar la función del ejercicio anterior),
   - para todas las posiciones pasadas en la lista existe una cruz en esa posición en el tablero de Tateti.
   Ejemplo, dado el siguiente Tateti:

   ```
   X O
     X
   O   X
   ```

   y la lista `[1,5,9]`, `litoCruz` debería devolver verdadero.

Ejercicio 8. Se cuenta con el tipo compuesto SimilYenga que modela un tablero de SimilYenga. Se busca implementar en Haskell algunos problemas sobre este tipo compuesto mediante su tipo algebraico que se define con los siguientes constructores:

```
data SimilYenga = Nuevo | Izq Int Yenga | Der Int Yenga
```

que permiten crear un SimilYenga nuevo (vacío), agregar una pieza a la izquierda (con cierto peso) o a la derecha (también con cierto peso).

1. `estaBalanceado :: SimilYenga -> Bool`, que devuelve verdadero sólo si se cumplen estas 2 condiciones a la vez:
   - la cantidad de piezas a izquierda es igual a la cantidad de piezas a derecha,
   - la suma de los pesos a izquierda y derecha coincide.

   Por ejemplo, aplicar `estaBalanceado` al SimilYenga `Izq 1 (Der 5 Nuevo)` debería devolver Falso ya que la suma de los pesos a cada lado difiere

2. `compactar :: SimilYenga -> SimilYenga`, que toma un SimilYenga y lo compacta de la siguiente manera:
   - Cada vez que haya dos piezas a izquiera o derecha consecutivas, las debe reemplazar por una sola, y el peso de ésta será la suma de los pesos de las piezas reemplazadas,
   Ejemplo, dado el siguiente SimilYenga: `Der 3(Izq 1 (Izq 5 Nuevo))`
   debería devolver `Der 3(Izq 6 Nuevo)`.