Programación Funcional

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todas las funciones y propiedades vistas en clase, aclarando la referencia. Cualquier otra función o propiedad que se utilice* ***debe ser definida o demostrada****.*
- *No se olvide de poner nombre, nro. de alumno, nro. de hoja y cantidad total de hojas en cada una de las hojas.*
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución.*
- *Recuerde que reusar código es una forma muy eficiente de disminuir el tiempo necesario para programar.*
- *La intención de la evaluación es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que sabe, explicar lo que se propone antes de escribir código y probar sus funciones con ejemplos.*

Versión preliminar

Considere la siguiente representación de polinomios con coeficientes enteros:

```
data Poli = Cte Int | Var | Add Poli Poli | Mul Poli Poli
```

Con la que por ejemplo el polinomio $P(x) = x^2 + 3x + 5$ puede representarse de la siguiente manera:

```
Add (Mul Var Var) (Add (Mul (Cte 3) Var) Cte 5)
```

Ejercicio 1

Escriba las siguientes funciones:

a) `eval :: Poli ->Int ->Int`
que retorna el resultado de evaluar un polinomio P con un valor x dado ($P(x)$).

b) `mEscalar :: Poli ->Int ->Poli`
que retorna el polinomio obtenido luego de multiplicar cada constante y variable por un valor entero.

c) `sOptimize :: Poli ->Poli`
que retorna un polinomio equivalente al tomado como parámetro pero donde las operaciones de suma y multiplicación entre constantes ya fueron resueltas, es decir, un polinomio en donde no existen constructores de la forma `Add (Cte _) (Cte _)` ni `Mul (Cte _) (Cte _)`

Ejercicio 2

Indique el tipo y de una implementación de una función que generalice la recursión sobre estos polinomios (`foldPoli`) y escriba las funciones del punto 1 utilizando esta función.

Ejercicio 3

Escriba las siguientes funciones sin utilizar recursión explícita, es decir, usando los esquemas de recursión de listas o listas por comprensión.

a) `evalAll :: Poli ->[Int] ->[Int]`
que para cada valor x de la lista de entrada devuelve la evaluación del polinomio $P(x)$ en la lista resultado.

b) `evalAllCoord :: Poli ->[Int] ->[(Int, Int)]`
que para cada valor x de la lista de entrada devuelve una tupla (x, P(x)) en la lista resultado.

c) `roots :: Poli ->[Int]`
que retorna una lista con los valores para los cuales evaluar el polinomio da 0. Indique si la función es parcial o total.

Ejercicio 4

Dada la siguiente función:

```
derive :: Poli ->Poli
derive (Cte n) = Cte 0
derive Var = Cte 1
derive Add p1 p2 = Add (derive p1) (derive p2)
derive Mul p1 p2 = Add (Mul (derive p1) p2) (Mul p1 (derive p2))
```

Considere sólo enteros positivos y demuestre por inducción estructural que:

$$\forall \texttt{p:Poli},\ \texttt{eval p 0 == 0} \Rightarrow \texttt{eval p 1} \leq \texttt{eval (derive p) 1}$$

Dado que trabaja sólo con enteros positivos puede utilizar las siguientes propiedades:

- $\forall x, y, x + y = 0 \Rightarrow x = 0 \wedge y = 0$
- $\forall x, y, x * y = 0 \Rightarrow x = 0 \vee y = 0$
- $\forall x, y, x \leq y \Rightarrow \forall z, x + z \leq y + z$