UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Parcial de Programación Funcional– 02/11/2009

LU:
Apellidos:
Nombres:

Aclaraciones: El parcial NO es a libro abierto. Cualquier decisión de interpretación que se tome debe ser aclarada y justificada. Para aprobar se requieren al menos 60 puntos. Entregar cada ejercicio en hoja separada. No está permitido utilizar alto orden. Al igual que para el TP, para la resolución del parcial, se pueden usar únicamente las funciones y operadores `last`, `init`, `head`, `tail`, `!!`, `reverse`, `++`, `elem`, `length` y los operadores de comparación entre elementos de un mismo tipo.

En el comienzo del parcial trabajaremos con los tipos de datos abstractos `Chofer`, `Micro` y `Empresa`, definidos de la siguiente manera:

`tipo` Nombre = String ;

`tipo Chofer` {
  `observador nombre` (c : Chofer) : Nombre ;
  `observador sueldo` (c : Chofer) : Float ;
  `invariante` $sueldo(c) \geq 0$;
}

`tipo Viaje` {
  `observador unidad` (v : Viaje) : Int ;
  `observador chofer` (v : Viaje) : Nombre ;
  `observador origen` (v : Viaje) : Nombre ;
  `observador destino` (v : Viaje) : Nombre ;
  `observador distancia` (v : Viaje) : Float ;
  `invariante` $origen(v) \neq destino(v)$;
  `invariante` $distancia(v) > 0$;
}

`tipo Empresa` {
  `observador nombre` (e : Empresa) : Nombre ;
  `observador choferes` (e : Empresa) : [Chofer] ;
  `observador viajes` (e : Empresa) : [Viaje] ;
  `invariante` $distintos(listaNombres(choferes(e)))$;
  `invariante` $(\forall v \leftarrow viajes(e))$
    $chofer(v) \in listaNombres(choferes(e))$;
}

`aux distintos` (l:[T]) : Bool $= (\forall i, j \leftarrow [0..|l|), i \neq j) l_i \neq l_j$;
`aux listaNombres` (l:[Chofer]) : [Nombre] $= [nombre(x) | x \leftarrow l]$;

Las funciones que implementan estos tipos en Haskell son `nombreC :: Chofer -> Nombre`, `sueldoC :: Chofer -> Float`, `unidadV :: Viaje -> Int`, `choferV :: Viaje -> Nombre`, `origenV :: Viaje -> Nombre`, `destinoV :: Viaje -> Nombre`, `distanciaV :: Viaje -> Float`
`nombreE :: Empresa -> Nombre`, `choferesE :: Empresa -> [Chofer]` y `viajesE :: Empresa -> [Viaje]`

Ejercicio 1. [35 puntos] Implementar en Haskell los siguientes problemas especificados más adelante

a) **[15 p.]** `problema esElDestino` (e : Empresa) = `result` : [(Nombre,[Nombre])]
b) **[20 p.]** `problema masCalle` (e : Empresa) = `result` : [Chofer]

`problema esElDestino` (e : Empresa) = `result` : [(Nombre,[Nombre])] {
  `asegura` $distintos([prm(r) | r \leftarrow result])$;
  `asegura` $(\forall r \leftarrow result) distintos(sgd(r))$;
  `asegura` $(\forall v \leftarrow viajes(e))(\exists r \leftarrow result, n \leftarrow sgd(r)) prm(r) == destino(v) \wedge chofer(v) == n$;
  `asegura` $(\forall r \leftarrow result, n \leftarrow sgd(r))(\exists v \leftarrow viajes(e)) prm(r) == destino(v) \wedge chofer(v) == n$;
}

`problema masCalle` (e : Empresa) = `result` : [Chofer] {
  `asegura` $mismos(result, [c | c \leftarrow choferes(e), (\forall x \leftarrow choferes(e)) recorrido(c, e) \geq recorrido(x, e)])$;
  `aux recorrido` (c:Chofer, e:Empresa) : Float $= \sum [distancia(v) | v \leftarrow viajes(e), chofer(v) == nombre(c)]$;
}

Tipos algebraicos.

<u>Nota Importante</u>: No está permitido utilizar el tipo lista para resolver los ejercicios de tipos algebraicos (Ejercicios 2 y 3).

Se cuenta con el tipo compuesto Recorrido que modela el recorrido de una tortuga. Básicamente, la tortuga parte de una posicón incial *(x,y)*. Luego puede ir moviéndose en el sentido Norte, Sur, Este u Oeste sólo de a un paso (las tortugas son muuuuuy lentas). Si la tortuga se mueve hacia el Norte, se suma uno a la coordenada *y*. Si se mueve hacia el Sur se le resta en uno la coordenada *y*. Al moverse hacia el Este se le suma en uno la coordenada *x*. Por último, al moverse hacia el Oeste se le resta en uno la coordenada *x*.

En definitiva, contamos con el tipo compuesto Recorrido
`tipo` PuntoCardinal = Norte, Sur, Este, Oeste ;

`tipo Recorrido` {
  `observador posicionInicial` (r: Recorrido) : (Int,Int) ;
  `observador movimientos` (r: Recorrido) : [PuntoCardinal] ;
}

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

```
data Recorrido = Inicio Int Int | Norte Recorrido | Sur Recorrido | Este Recorrido | Oeste Recorrido
```

que permiten ubicar en una posición inicial x y (Inicio Int Int) a la tortuga y luego moverse un paso al Norte, Sur, Este u Oeste respectivamente.

Por ejemplo, un recorrido de 4 movimientos podría ser:
`Sur (Oeste (Norte (Norte (Inicio 1 2))))`.

Ejercicio 2. [10 puntos] Implementar el problema `posicionActual :: Recorrido -> (Int,Int)`

```
problema posicionActual (r : Recorrido) = result : (ℤ,ℤ)  {
  asegura result == acum(sumaMovida(m, p) | p : (ℤ, ℤ) = posicionInicial(r), m ← movimientos(r)) ;
  aux sumaMovida (m:PuntoCardinal, p:(ℤ,ℤ)) :
     p:(ℤ,ℤ) = (prm(p) + 1 ∗ β(m == Este) − 1 ∗ β(m == Oeste), sgd(p) + 1 ∗ β(m == Norte) − 1 ∗ β(m == Sur)) ;
}
```

Ejemplo:
`Inicio 1 2`.debería devolver (1,2)
`Norte (Inicio 1 2)`.debería devolver (1,3)
`Sur (Inicio 1 2)`.debería devolver (1,1)
`Sur (Oeste (Norte (Norte (Inicio 1 2))))`.debería devolver (0,3)

Ejercicio 3. [35 puntos] Implementar `seCruzan:: Recorrido -> Recorrido -> Bool`, que dado el recorrido de la tortuga Manuelita y un tortugo que pasó devuelve verdadero si y sólo si en algun momento ambos recorridos pasan por la misma posición.

```
problema seCruzan (r1,r2 : Recorrido) = result : Bool  {
  asegura result == (∃p1 ← posiciones(r1), p2 ← posiciones(r2))p1 == p2 ;
  aux posiciones (r:Recorrido) : [(ℤ,ℤ)] =
     acum(ps + +sumaMovida(m, ps|ps|−1) | ps : [(ℤ, ℤ)] = [posicionInicial(r)], m ← movimientos(r)) ;
  aux sumaMovida (m:PuntoCardinal, p:(ℤ,ℤ)) :
     p:(ℤ,ℤ) = (prm(p) + 1 ∗ β(m == Este) − 1 ∗ β(m == Oeste), sgd(p) + 1 ∗ β(m == Norte) − 1 ∗ β(m == Sur)) ;
}
```

Ejemplo: Si contamos con los siguientes recorridos:

recorridoA =`Inicio 1 2`
recorridoB =`Norte (Inicio 1 2)`
recorridoC =`Este (Norte (Inicio 0 0))`
recorridoD =`Oeste (Oeste (Sur (Inicio 2 2)))`

entonces:

`seCruzan recorridoA recorridoB` es verdadero (porque ambos parten de la misma posición inicial, se encuentran allí).
`seCruzan recorridoA recorridoC` es falso. (nunca se cruzan)
`seCruzan recorridoC recorridoD` es verdadero (se cruzan en (0,1) y en (1,1))

Ejercicio 4. [20 puntos] (Ejercicio 3 tomado de la práctica 8)
Se cuenta con el tipo compuesto *Polinomio* (de coeficientes enteros), definido de la siguiente manera:

```
tipo Polinomio{
   observador grado(p: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:

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

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