UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Recuperatorio Parcial de Programación Funcional– 12/12/2008

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.

En el comienzo del parcial trabajaremos con los tipos de datos abstractos `Peon`, `Vaca` y `Tambo`, definidos de la siguiente manera:

```
tipo Peon {
  observador nombre (p : Peon) : String ;
  observador sueldo (p : Peon) : Float ;
  invariante sueldo(p) ≥ 0 ;
}

tipo Vaca {
  observador nombre (v : Vaca) : String ;
  observador litros (v : Vaca) : Float ;
  observador costo (v : Vaca) : Float ;
  observador aCargo (v : Vaca) : String ;
  invariante litros(v) ≥ 0 ∧ costo(v) ≥ 0 ;
}
```

```
tipo Tambo {
  observador razonSocial (t : Tambo) : String ;
  observador peones (t : Tambo) : [Peon] ;
  observador vacas (t : Tambo) : [Vaca] ;
  invariante distintos(peones(t)) ;
  invariante distintos(vacas(t)) ;
  invariante (∀v ← vacas(t))aCargo(v) ∈ listaNombres(peones(t)) ;
  aux distintos (l:[T]) : Bool = (∀i, j ← [0..|l|), i ≠ j)nombre(l_i) ≠ nombre(l_j) ;
  aux listaNombres (l:[T]) : [String] = [nombre(x)|x ← l] ;
}
```

Las funciones que implementan estos tipos en Haskell son `nombreP :: Peon -> String`, `sueldoP :: Peon -> Float`, `nombreV :: Vaca -> String`, `litrosV :: Vaca -> Float`, `costoV :: Vaca -> Float`, `aCargoV :: Vaca -> String`. `razonSocialT :: Tambo -> String`, `peonesT :: Tambo -> [Peon]` y `vacasT :: Tambo -> [Vaca]`

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

a) **[20 p.]** `problema milka` (ts : [Tambo]) = `result` : [Vaca]
b) **[15 p.]** `problema laCasaGana` (precioLitro: Float, ts : [Tambo]) = `result` : [(String,Float)]

```
problema milka (ts: [Tambo]) = result : [Vaca] {
  requiere distintos([ nombre(v) | v ← todasLasVacas(ts) ]) ;
  asegura mismos(result, lasQueMenosCuestan(lasQueMasProducen(todasLasVacas(ts)))) ;
  aux todasLasVacas (ts:[Tambo]) : [Vaca] = [ v | t ← ts, v ← vacas(t) ]) ;
  aux distintos (l:[T]) : Bool = (∀i, j ← [0..|l|), i ≠ j)l_i ≠ l_j ;
  aux lasQueMasProducen (vs:[Vaca]) : [Vaca] = [ v | v ← vs, (∀c ← vs)litros(v) ≥ litros(c) ] ;
  aux lasQueMenosCuestan (vs:[Vaca]) : [Vaca] = [ v | v ← vs, (∀c ← vs)costo(v) ≤ costo(c) ] ;
}

problema laCasaGana (precioLitro: Float, ts : [Tambo]) = result : [(String,Float)] {
  asegura mismos(result, [ (razonSocial(t), gananciaVacas(precioLitro, t) − costoPeones(t)) | t ← ts ]) ;
  aux gananciaVacas (precioLitro:Float, t:Tambo) : Float = ∑[ litros(v) ∗ precioLitro − costo(v) | v ← vacas(t) ] ;
  aux costoPeones (t:Tambo) : Float = ∑[ sueldo(p) | p ← peones(t) ] ;
}
```

Tipos algebraicos.

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

Se cuenta con el tipo compuesto Libro en Construcción que modela un libro que no está terminado aun.
Tipo NroPagina = Int
Tipo Texto = String

```
tipo LibroEnConstruccion {
  observador paginas (l:LibroEnConstruccion) : [NroPagina] ;
  observador texto (l:LibroEnConstruccion, n:NroPagina) : Texto ;
      requiere n ∈ paginas(l) ;
  invariante distintos(paginas(l)) ;
  aux distintos (l:[T]) : Bool = (∀i, j ← [0..|l|), i ≠ j)l_i ≠ l_j ;
}
```

La idea general es que, dado un libro en construcción, se puede listar las páginas y acceder a su contenido.

Se busca implementar en Haskell algunos problemas sobre el tipo compuesto LibroEnConstruccion mediante el tipo algebraico de mismo nombre. Dicho tipo está definido con los siguientes constructores

Tipo NroPagina = Int
Tipo Texto = String

data LibroEnConstruccion = Nuevo | Pagina NroPagina Texto LibroEnConstruccion

que permiten crear un LibroEnConstruccion "Nuevo" y agregar una página con su texto a un LibroEnConstruccion. Por ejemplo, un LibroEnConstruccion que tiene 2 páginas podría ser:

```
Pagina 26 "pero como no le hizo caso, ..." (Pagina 8 "Y asi fue como ..." Nuevo).
```

Como restricción para los ejercicios que se detallan a continuación vamos a suponer que un LibroEnConstruccion no puede contener dos veces el mismo NroPagina. Es decir, este caso que se muestra a continuación no es posible ya que posee 2 veces la página 8:

```
Pagina 8 "pero como no le hizo caso, ..." (Pagina 8 "Y asi fue como ..." Nuevo).
```

Ejercicio 2. [**20 puntos**] Implementar `consolidar :: LibroEnConstruccion-> LibroEnConstruccion -> LibroEnConstruccion`, que toma dos libros, cada uno de ellos con sus páginas en orden, y devuelve un nuevo libro con las páginas de ambos y en orden. En el caso que tengan alguna página en común, esas página se deben agregar una sola vez y con el texto de ambas concatenado (es indistinto qué texto va primero). Ejemplo:

Si el primer libro es `Pagina 9 "Texto 9A" (Pagina 8 "Texto 8" (Pagina 2 "Texto 2" Nuevo))`.
y el segundo libro es `Pagina 9 "Texto 9B" (Pagina 5 "Texto 5" (Pagina 4 "Texto 4" Nuevo))`.
el resultado de aplicarles consolidar, debería retornar:
`Pagina 9 "Texto 9ATexto 9B" (Pagina 8 "Texto 8" (Pagina 5 "Texto 5" (Pagina 4 "Texto 4" (Pagina 2 "Texto 2" Nuevo))))`.

Ejercicio 3. [**25 puntos**] Implementar `esPlagio :: LibroEnConstruccion->LibroEnConstruccion->Bool`, que toma dos libros y devuelve True solamente en el caso que más de la mitad de las páginas del primer libro se repiten en el segundo. Un página se considera repetida si el texto completo de esa página se encuentra en otra página.

Ejercicio 4. [**20 puntos**] (Ejercicio tomado de la práctica 7) *capicua* :: $[Char] \rightarrow bool$, que devuelve verdadero si la lista es capicua. Por ejemplo *capicua* $[a, c, b, b, c, a]$ es $true$, *capicua* $[a, c, b, d, a]$ es $false$.