UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Recuperatorio Parcial Programación Funcional– 14/12/2009

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.

Rememoranro exámenes pasados, en el país de Liliput, un grupo de personitas decide organizar la Comisión Federal de Diariodifusión (ComFeD). El ComFeD es un organismo autárquico del Estado responsable de regular, controlar y fiscalizar la creación y funcionamiento de diarios en todo el país. El ComFeD monitorea todos los diarios. Los diarios están compuestos por secciones, donde cada sección cuenta con variadas noticias. Además, el ComFeD lleva registro de quién es dueño de cada diario. También, aunque no lo crean, tiene la capacidad de analizar si una noticia de un diario determinado tiene tendencia oficialista, moderada u opositora. De esta manera, obtenemos la siguiente representación:

`tipo` Nombre = String ;
`tipo` Seccion = Policiales, Política, Deportes, Espectáculos ;
`tipo` Tendencia = Oficialista, Moderado, Opositor ;

`tipo Noticia` {
  `observador autor` (n :Noticia) : Nombre ;
  `observador informacion` (n :Noticia) : String ;
}

`tipo Diario` {
  `observador nombre` (d :Diario) : Nombre ;
  `observador secciones` (d :Diario) : [Seccion] ;
  `observador noticias` (d :Diario, s :Seccion) : [Noticia] ;
    `requiere seccionDelDiario` : $s \in secciones(d)$;
  `invariante sinSecRepes` : $sinRepetidos(secciones(d))$;
  `invariante sinNotRepes` : $sinNoticiasRepetidas(d)$;
}

`aux sinRepetidos` (l : [T]) : Bool = $(\forall i \leftarrow [0..|l|))\, l_i \notin l(i..|l|)$;
`aux sinNoticiasRepetidas` (d:Diario) : Bool = $sinRepetidos([n|s \leftarrow secciones(d), n \leftarrow noticias(d,s)])$;

`tipo ComFeD` {
  `observador diarios` (c :ComFeD) : [Diario] ;
  `observador analisis` (c :ComFeD, dn :Nombre, nt :Noticia) : Tendencia ;
    `requiere diarioDelComFeD` : $diarioDelComFeD(c, dn)$;
    `requiere noticiaDelDiario` : $(\exists d \leftarrow diarios(c), s \leftarrow secciones(d), nombre(d) == dn) nt \in noticias(d,s)$;
  `observador dueño` (c :ComFeD, dn :Nombre) : Nombre ;
    `requiere diarioDelComFeD` : $diarioDelComFeD(c, dn)$;
  `invariante noHayDiariosRepetidos` : $sinRepetidos(nomDiarios(c))$;
}

`aux nomDiarios` (c :ComFeD) : [Nombre] = $.[nombre(d)|d \leftarrow diarios(c)]$;
`aux diarioDelComFeD` (c :ComFeD, n :Nombre) : Bool = $n \in nomDiarios(c)$;

Las funciones que implementan estos tipos en Haskell son `diariosC :: ComFeD -> [Diario]`, `analisisC :: ComFeD -> Nombre -> Noticia -> Tendencia`, `duenoC :: ComFeD -> Nombre -> Nombre`, `nombreD :: Diario -> Nombre`, `seccionesD :: Diario -> [Seccion]`, `noticiasD :: Diario -> Seccion -> [Noticia]`, `autorN :: Noticia -> Nombre`, `informacionN :: Noticia -> String`

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

a) **[15 p.]** `problema plural` (c : ComFeD) = `result` : [Diario]
b) **[20 p.]** `problema esOpositor` (c : ComFeD, n:Nombre) = `result` : Bool

`problema plural` (c : ComFeD) = `result` : [Diario] {
  `asegura` $mismos(result, [d|d \leftarrow diarios(c), (\forall x \leftarrow diarios(c)) cantAutoresDistintos(d) \geq cantAutoresDistintos(x)])$;
  `aux cantAutoresDistintos` (d :Diario) : $\mathbb{Z} = |sacarRepe([autor(nt)|s \leftarrow secciones(d), nt \leftarrow noticias(d,s)])|$;
  `aux sacarRepe` (l : [T]) : [T] = $[l_i|i \leftarrow [0..|l|), l_i \notin l(i..|l|)]$;
}

`problema esOpositor` (c : ComFeD, n:Nombre) = `result` : Bool {
  `asegura` $result == |analisisAutor(c,n)| \geq 1 \wedge (\forall t \leftarrow analisisAutor(c,n)) t == Opositor$;
  `aux analisisAutor` (c :ComFeD, a:Nombre) : [Tendencia] = $[analisis(c, nombre(d), nt)|$
    $d \leftarrow diarios(c), s \leftarrow secciones(d), nt \leftarrow noticias(d,s), autor(nt) == a]$;
}

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

Se cuenta con el tipo compuesto Movilizacion como herramienta para la organización de movilizaciones sociales. Este tipo compuesto administra micros y personas. Dado que los micros suelen no poseer patente, se decidió numerarlos.

Dada una movilización, a través de sus observadores, se puede obtener la lista de micros que participan de la misma, la capacidad máxima de cada micro y la cantidad de personas asignadas que cada uno de ellos posee.

En definitiva, contamos con el tipo compuesto Movilizacion:

`tipo` Cantidad = $\mathbb{Z}$;
`tipo` NroMicro = $\mathbb{Z}$;

`tipo Movilizacion` {
  `observador micros` (m: Movilizacion) : [NroMicro] ;
  `observador capacidadMaxima` (m: Movilizacion, n: NroMicro) : Cantidad ;
      `requiere` $n \in micros(m)$;
  `observador personasAsignadas` (m: Movilizacion, n: NroMicro) : Cantidad ;
      `requiere` $n \in micros(m)$;
  `invariante` $(\forall i \leftarrow micros(m))0 \leq capacidadMaxima(m, i)$;
  `invariante` $(\forall i \leftarrow micros(m))0 \leq personasAsignadas(m, i) \leq capacidadMaxima(m, i)$;
  `invariante` $sinRepetidos(micros(m))$;
}

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

```
data Mov = NuevaMov | AgregarMicro Int Int Mov | AbordarMicro Int Mov | BajarseDelMicro Int Mov
```

La idea es que dada una nueva movilización se pueda agregar micros con su respectivo número identificatorio (primer parámetro) y su capacidad máxima (segundo parámetro) y subir o bajar una persona a un micro determinado (donde el primer parámetro corresponde al número identificatorio del micro).

Por ejemplo, una movilización a la que asisten 2 micros (que numeramos 4 y 7, y que tienen una capacidad máxima de 10 personas), en el que al primero (al numerado como 4) se suben 2 personas y al segundo (numerado como 7) una sola, se podría modelar de la siguiente manera:

```
AbordarMicro 7 (AgregarMicro 7 10 (AbordarMicro 4 (AbordarMicro 4 (AgregarMicro 4 10 NuevaMov))))
```

Es imortante notar que existen distintas maneras de modelar este mismo escenario, por ejemplo, una manera alternativa podría ser la siguiente:

```
AbordarMicro 7 (AbordarMicro 4 (AgregarMicro 7 10 (AbordarMicro 4 (AgregarMicro 4 10 NuevaMov))))
```

Ejercicio 2. [45 puntos] Implementar el problema `bienFormado :: Mov -> Bool` , que devuelve verdadero si y sólo si se cumplen las siguiente condiciones a la vez:

- siempre que un pasajero aborda un micro, el micro ha sido agregado previamente
- a medida que los pasajeros suben o bajan del micro nunca superan la capacidad del mismo

Ejemplo:

`bienFormado (AgregarMicro 4 10 (AbordarMicro 4 NuevaMov))`.Da Falso porque la persona aborda el micro antes de que sea agregado.

`bienFormado BajarseDelMicro 4 (AbordarMicro 4 (AbordarMicro 4 (AgregarMicro 4 1 NuevaMov)))`.Da Falso porque si bien la cantidad de pasajeros arriba del micro es 1, en un momento fueron 2 y superaron la capacidad máxima del micro que es 1.

Ejercicio 3. [20 puntos] (Ejercicio 6.4 tomado de la práctica 7) Implementar la función *descomponerEnPrimos* :: [*Int*] → [[*Int*]], que devuelve la lista de listas, que resulta de descomponer en números primos cada uno de los números de la lista original. Por ejemplo *descomponerEnPrimos* $[2, 10, 6, 4]$ es $[[2], [2, 5], [2, 3], [2, 2]]$.