UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Parcial de Programación Funcional– 09/10/2013

LU:
Apellidos:
Nombres:
Orden:
Turno:

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.

Instituto Nacional Grande por las Amistades Libres y Luminosas Sostenidas (*Ingalls*) es un instituto que busca generar felicidad en la gente. En particular de sus propios miembros.
Es por esto que le importa qué tan feliz es una persona cada día de la semana y actúa en consecuencia.
Cada persona tiene un nombre y, para cada día de la semana, su estado de ánimo.
A su vez las personas ingresan al *Ingalls* cuando son evangelizados por un miembro del mismo.
De esta manera, obtenemos la siguiente representación:

```
tipo Nombre = String;
tipo NivelDeFelicidad = ℤ;
tipo Día = Domingo, Lunes, Martes, Miércoles, Jueves, Viernes, Sábado;
tipo Ánimo = [(Día, NivelDeFelicidad)];

tipo Persona {
  observador nombre (p : Persona) : Nombre;
  observador animos (p : Persona) : Ánimo;
  invariante animosPositivosYNumerados :
    (∀a ← animos(p))sgd(a) ≥ 1 ∧ sgd(a) ≤ 10;
  invariante sinRepetidoDiaYDiasCompletos :
    sinRepetidos(primeros(animos(p)) ∧ |animos(p)| == 7;
}

aux sinRepetidos (l : [T]) : Bool = (∀i ← [0..|l|)) l_i ∉ l(i..|l|);
aux nombres (l : [Persona]) : [Nombre] = [nombre(x)|x ← l];
aux primeros (l : [(A,B)]) : [A] = [prm(x)|x ← l];
aux segundos (l : [(A,B)]) : [B] = [sgd(x)|x ← l];
aux promedio (l: [ℤ]) : ℤ = sum(l)/|l|;
```

```
tipo Ingalls {
  observador integrantes (ing : Ingalls) : [Persona];
  observador evangelizo (ing : Ingalls, p: Persona) : [Persona];
      requiere p ∈ integrantes(i);
  invariante noHayTocayos : sinRepetidos(nombres(integrantes(ing)));
  invariante noSobreevangelizan : (∀p1 ← integrantes(ing))(∀p2 ←
    integrantes(p))p1 ≠ p2 → sinRepetidos(evangelizo(ing, p1)+
    +evangelizo(ing, p2));
  invariante noHayInfelices : (∀p ← integrantes(ing))(∀a ←
    animos(p))sgd(a) > 4;
}
```

Las funciones que implementan estos tipos en Haskell son `nombreP:: Persona -> Nombre`,
`animoP:: Persona -> Animo`,
`integrantesI :: Ingalls -> [Persona]`
`evangelizoI:: Ingalls -> Persona -> [Persona]`

Los tipos utilizados serán análogos en Haskell a los especificados.

Ejercicio 1. [40 puntos]

Implementar en Haskell los siguientes problemas especificados más adelante

a) [15 p.] `problema promediosDias (ing : Ingalls) = result : [(Dia, ℤ)]`
b) [15 p.] `problema buenaIncorporacion (ing : Ingalls, p : Persona) = result : [Persona]`
c) [10 p.] `problema superFeliz (ing: Ingalls) = result : Persona`

```
problema promediosDias (ing : Ingalls) = result : [(Dia, ℤ)] {
  asegura |result| == 7;
  asegura sinRepetidos(primeros(result));
  asegura (∀prom ← result)promedio(felicidades(ing, sgd(prom))) == prm(prom);
}
aux felicidades (ing: Ingalls, d: Día) : [ℤ] = [prm(a)|p ← integrantes(ing), a ← animos(p), sgd(a) == d];

problema buenaIncorporacion (ing : Ingalls, p: Persona) = result : [Persona] {
  requiere p ∈ integrantes(ing);
  asegura (∀evang ← result)(evang ∈ evangelizo(ing, p) ∧ maxFelicidad(p) < maxFelicidad(evang));
  asegura (∀evang ← evangelizo(ing, p))(maxFelicidad(p) < maxFelicidad(evang)) → evang ∈ result;
  asegura sinRepetidos(result);
}
aux maxFelicidad (p: Persona) : ℤ = [f1|f1 ← segundos(animos(p)), (∀f2 ← segundos(animos(p))f1 ≥ f2)]0;

problema superFeliz (ing: Ingalls) = result : Persona {
  requiere |integrantes(ing)| > 0;
```

asegura $result \in integrantes(ing)$;
asegura $(\forall p \leftarrow integrantes(ing))promedioFelicidad(p) \leq promedioFelicidad(result)$;
}
aux promedioFelicidad (p: Persona) : $\mathbb{Z} = promedio([sgd(a)|a \leftarrow animos(p)])$;

Tipos algebraicos. ⇒ Nota Importante: No está permitido CONVERTIR EL TIPO ALGEBRAICO AL TIPO LISTAS para resolver los ejercicios de tipos algebraicos (Ejercicios 2 y 3).

Se cuenta con el tipo compuesto HuyQueTimba que modela los movimientos de una reconocida empresa bursátil.

```
tipo Operacion = Compra, Venta;
tipo Monto = Int;
tipo Moviemento = (Operacion, Monto);
tipo HuyQueTimba {
  observador movimientos (h: HuyQueTimba) : [Movimiento];
  observador saldoInicial (h: HuyQueTimba) : Monto;
  invariante noQuedoEnRojo(h);
  invariante soloMovimientosPositivos(h);
}
aux noQuedoEnRojo (h:HuyQueTimba) : Bool = (∀i ← [0..|movimientos(h)| − 1])sumaHasta(i, h) + saldoInicial(h) ≥ 0;
aux soloMovimientosPositivos (h:HuyQueTimba) : Bool = (∀mov ← [0..|movimientos(h)| − 1])sgd(mov) ≥ 0;
aux valor (m: Movimiento) : Z = ifThenElse(prm(m) == Compra, −1 ∗ sgd(m), sgd(m));
aux sumaHasta (h:HuyQueTimba, i: Z) : Z = sum([valor(movimientos(h)_k)|k ← [0..i]]);
```

donde el observador movimientos(h) devuelve las lista de movimientos de la empresa en el orden en que fueron ejecutados, saldoInicial(h) indica el saldo con el que comienza la empresa y soloMovimientosPositivos(h) indica que todos los montos son positivos (o cero).

Se busca implementar en Haskell algunos problemas sobre el tipo compuesto HuyQueTimba mediante los tipos algebraicos `Operacion` y `HuyQueTimba`. Dichos tipos se definen a través de los siguientes constructores:

```
data Operacion = Compra | Venta deriving (Eq)
  data HuyQueTimba = Saldo Int| HQT Operacion Int HuyQueTimba deriving (Eq)
```

donde en el tipo HuyQueTimba, el primer constructor permite construir una nueva empresa y el segundo hacer una operacion. Por ejemplo, una empresa h que compra y vende con la siguiente secuencia $movimientos(h) == [[Vende, 10], [Vende, 5], [Compra, 2]]$ se construiría de la siguiente manera (Suponiendo saldo inicial de 8): `HQT Compra 2 (HQT Vende 5 (HQT Vende 10 (Saldo 8)))`.

Ejercicio 2. [25 puntos]

Implementar `queCasualidadCompramosSimilar:: HuyQueTimba -> HuyQueTimba -> Int -> Bool` , que a partir de dos HuyQueTimba, devuelve true si tienen al menos una secuencia de compras iguales de longitud mayor o igual al Int pasado como parámetro.

Por ejemplo:

```
queCasualidadCompramosSimilar (HQT Compra 2 (HQT Vende 5
(HQT Vende 10 (Saldo 8)))) (HQT Compra 2 (HQT Vende 10 (Saldo 8))) 2
```

debería dar como resultado *True* pero

```
queCasualidadCompramosSimilar (HQT Compra 2 (HQT Vende 5
(HQT Vende 10 (Saldo 8)))) (HQT Compra 2 (HQT Vende 10 (Saldo 8))) 3
```

debería dar como resultado *False*.

Ejercicio 3. [20 puntos]

Implementar `eliminaOperacionesAutomaticas:: HuyQueTimba -> HuyQueTimba`, que a partir de un HuyQueTimba, devuelve otro pero eliminando apariciones de operaciones del mismo tipo y que son ascendentes o descendentes consecutivas.

Por ejemplo:

```
eliminaOperacionesAutomaticas(HQT Compra 5(HQT Compra 4 (HQT Compra 3  (HQT Compra 4(HQT Compra 3 (Saldo 8)))))
```

debería dar como resultado

```
Saldo 8
```

y

```
eliminaOperacionesAutomaticas(HQT Compra 4 (HQT Venta 3 (HQT Compra 10  (Saldo 8))))
```

debería dar como resultado

```
HQT Compra 4 (HQT Venta 3 (HQT Compra 10 (Saldo 8)))
```

y

```
eliminaOperacionesAutomaticas(HQT Compra 4 (HQT Venta 4 (HQT Venta 3  (HQT Venta 2 (Saldo 8)))))
```

debería dar como resultado

```
HQT Compra 4 ((Saldo 8))
```

Ejercicio 4. [15 puntos]

(Ejercicio tomado de la práctica 7) *descomponerEnPrimos* :: $[Int] \rightarrow [[Int]]$, 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* $[2, 12, 6]$ es $[[2], [2, 2, 3], [2, 3]]$. Nota: no importa el orden de cada lista de primos.