UBA – Facultad de Ciencias Exactas y Naturales – Departamento de Computación– Algoritmos y Estructura de Datos I
Recuperatorio del Parcial de Programación Funcional– 04/12/2013

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.
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 edad (p : Persona) : ℤ;
  observador animos (p : Persona) : Ánimo;
  invariante animosPositivosYNumerados :
    (∀a ← animos(p))sgd(p) ≥ 1 ∧ sgd(p) ≤ 10;
  invariante sinRepetidoDiaYDiasCompletos :
    sinRepetidos(primeros(animos(p)) ∧ |animos(p)| == 7;
  invariante sonMayores : edad(p) > 18;
}
aux sinRepetidos (l : [T]) : Bool = (∀i ← [0..|l|)) l_i ∉ l(i..|l|);
aux nombres (l : [Persona]) : [A] = [nombre(x)|x ← l];
aux primeros (l : [(A,B)]) : [A] = [prm(x)|x ← l];
```

Las funciones que implementan estos tipos en Haskell son

```
nombreP:: Persona -> Nombre,
animoP:: Persona -> Animo,
integrantesI :: Ingalls -> [Persona]
evangelizoI:: Ingalls -> [Persona] -> [Persona]
```

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

Ejercicio 1. [45 puntos]
Implementar en Haskell los siguientes problemas especificados más adelante

a) [20 p.] `problema evangelizadosDeEvangelizados (i : Ingalls, p:Persona) = result : [Persona]`
Devuelve los que fueron evangelizados por la persona pasada como parámetro y por cualquiera evangelizada por sus evangelizados, directa o indirectamente. Requiere que la persona exista en el Ingalls y devuelve una lista sin repetidos.
b) [10 p.] `problema choreoEvangelizacion (i : Ingalls, p: Persona) : Ingalls`
c) [15 p.] `problema viejosEvangelistas (i: Ingalls) = result : Persona`

```
problema choreoEvangelizacion (i : Ingalls, p:Persona) = result : Ingalls {
  requiere (∃p1 ← integrantes(i))|evangelizo(i, p1)| > 0;
  requiere p ∉ integrantes(i);
  asegura mismos(integrantes(result), p : integrantes(i));
  asegura (∀p1 ← integrantes(i), |evangelizo(i, p1)| < maxEvangelizados(i))
    mismos(evangelizo(i, p1), evangelizo(result, p1));
  asegura (∀p1 ← integrantes(result), |evangelizo(i, p1)| = maxEvangelizados(i))
    evangelizo(result, p1) == [];
  asegura (∀p1 ← integrantes(i), |evangelizo(i, p1)| = maxEvangelizados(i))
    evangelizo(i, p1) ⊆ evangelizo(result, p);
  asegura (∀p1 ← evangelizo(result, p))(∃p2 ← integrantes(i), |evangelizo(i, p2)| == maxEvangelizados(i))
    p1 ∈ evangelizo(i, p2);
}
aux maxEvangelizados (i: Ingalls) : ℤ = ||evangelizo(i, p1)||
  p1 ← integrantes(i), (∀p2 ← integrantes(i)|evangelizo(i, p1)| ≥ |evangelizo(i, p2)||0;
```

```
problema viejosEvangelistas (i: Ingalls) = result : Persona {
  requiere (∃p1 ← integrantes(i))|evangelizo(i, p1)| > 0;
  asegura result ∈ integrantes(i);
  asegura (∀p ← integrantes(i))|filtrarEdad(evangelizo(i, result), mayorEdad(i))| ≥ |filtrarEdad(evangelizo(i, p), mayorEdad(i
}
aux mayorEdad (i: Ingalls) : ℤ = [edad(p1)|
  p1 ← integrantes(i), (∀p2 ← integrantes(i)edad(p1) ≥ edad(p2)]0;
aux filtrarEdad (ps: [Persona], edad: ℤ) : [Persona] = [p|
  p ← ps, edad(p) ≥ edad];
```

Tipos algebraicos. ⇒ Nota Importante: No está permitido UTILIZAR EL TIPO LISTA para resolver los ejercicio de tipos algebraicos (Ejercicios 2 y 3).

Se cuenta con el tipo compuesto UyQueTimba que modela los movimientos reconocida empresa bursátil.

```
tipo Operacion = Compra, Venta;
tipo Monto = ℤ;
tipo Moviemento = (Operacion, Monto);
tipo UyQueTimba {
  observador movimientos (h: UyQueTimba) : [Movimiento];
  observador saldoInicial (h: UyQueTimba) : ℤ;
  invariante noQuedoEnRojo(h);
}
aux noQuedoEnRojo (h:UyQueTimba) : Bool = (∀i ← [0..|movimientos(h)|))sumaHasta(i, h) + saldoInicial(h) ≥ 0;
aux valor (m: Movimiento) : ℤ = ifThenElse(prm(m) == Compra, −1 ∗ sgd(m), sgd(m));
aux sumaHasta (h:UyQueTimba, 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 y saldoInicial(h) indica el saldo con el que comienza la empresa.

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

```
data Operacion = Compra | Venta deriving (Eq)
  data UyQueTimba = Saldo Nat | UQT Operacion Nat deriving (Eq)
```

donde en el tipo UyQueTimba, 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): `UQT Compra 2 (UQT Vende 5 (UQT Vende 10 (Saldo 8)))`.

Ejercicio 2. [15 puntos]

Implementar `evitarRojoReordeno:: UyQueTimba -> UyQueTimba`, que a partir de un UyQueTimba devuelve otro en el cual primero se hacen todas las ventas y luego todas las compras.

Por ejemplo:

`evitarRojoReordeno (UQT Venta 5 (UQT Compra 4 (UQT Compra 1 (Saldo 8)))`
debería dar como resultado `(Compra 4 (Compra 1 (Venta 5 (Saldo 8))))` o `(Compra 1 (Compra 4 (Venta 5 (Saldo 8))))`

```
problema evitarRojoReordeno (UQT: UyQueTimba) = result : UyQueTimba {
  asegura mismos(movimientos(result), movimientos(UQT));
  asegura ventasAntesDeCompras(primeros(result));
}
```

```
aux ventasAntesDeCompras (ops: [Operacion]) : Bool = (∀i ← [0..|ops| − 1))(opsi = COMPRA) → (∀j ← (i..|ops|))opsj ≠ VENTA;
```

Ejercicio 3. [25 puntos]

Implementar `maximoVoyYVuelvo:: UyQueTimba -> nat`, que devuelve el tamaño de la mayor subsecuencia intercalada de operaciones del UQT pasado como parámetro.

Por ejemplo:

`maximoVoyYVuelvo(UQT Compra 2 (UQT Vende 5 (UQT Compra 1 (UQT Vende 10 (UQT Compra 2(UQT Compra 5 (UQT Vende 1 ( Saldo 8)))))))))`
debería dar como resultado 5.

y

`maximoVoyYVuelvo(UQT Compra 2 (HQT Vende 5 (HQT Vende 2(HQT Compra 5 (Vende 6(Compra 2 (Compra 1( Saldo 8)))))))))`
debería dar como resultado 4.

```
problema maximoVoyYVuelvo (UQT: UyQueTimba) = result : ℤ {
  requiere movimientos(UQT) ≥ 2;
  requiere (∃i ← [0..|movimientos(UQT)| − 1))movimientos(UQT)i ≠ movimientos(UQT)i+1;
```

asegura $(\exists i \leftarrow [0..|movimientos(UQT)|), \exists j \leftarrow (i..|movimientos(UQT)|))intercalados(movimientos(UQT), i, j) \wedge$
$((\forall i2 \leftarrow [0..|movimientos(UQT)|), (\forall j2 \leftarrow (i2..|movimientos(UQT)|)))$
$intercalados(movimientos(UQT), i2, j2) \rightarrow (j - i) \geq (j2 - i2) \wedge (j - i) = result)$;

`x intercalados` (ls: [(Operacion, Monto)], i, j:$\mathbb{Z}$) : Bool $= (\forall k \leftarrow [i..j))prm(ls_k) \neq prm(ls_{k+1})$;

Ejercicio 4. [15 puntos]
(Ejercicio 2.8 tomado de la práctica 7)
plementar `aplanarConNBlancos :: [[Char]] -> Integer -> [Char]`, que a partir de una lista de palabras, arma una lista c
acteres concatenándolas e insertando n blancos entre cada par (n debe ser no negativo).