Clase de Pociones

Se pide desarrollar un programa Haskell que ayude a analizar las pociones que se enseñan a los alumnos en el colegio Hogwarts de Magia y Hechicería, y los efectos que pueden hacer sobre personas.
Estas son algunas funciones ya definidas:

```
aplicar3 f (a,b,c) = (f a, f b, f c)
invertir3 (a,b,c) = (c,b,a)
fst3 (a,_,_) = a
snd3 (_,b,_) = b
trd3 (_,_,c) = c

sinRepetidos [] = []
sinRepetidos (x:xs)
     | elem x xs = sinRepetidos xs
     | otherwise = x : sinRepetidos xs

maximoF _ [ x ] = x
maximoF  f ( x : y : xs)
     | f x > f y = maximoF f (x:xs)
     | otherwise = maximoF f (y:xs)
```

La información con la que se trabajará es la siguiente:

```
personas =
     [("Harry",(11, 5, 4)), ("Ron",(6,4,6)), ("Hermione",(8,12,2)), ("Draco",(7,9,6))]
```

Donde cada tupla de la lista tiene la forma:
("Nombre del mago o bruja", (nivelSuerte, nivelPoderDeConvencimiento, nivelFuerzaFisica))

```
f1 (ns,nc,nf) = (ns+1,nc+2,nf+3)
f2 = aplicar3 (max 7)
f3 (ns,nc,nf)
     | ns >= 8 = (ns,nc,nf+5)
     | otherwise = (ns,nc,nf-3)

misPociones = [
     ("Felix Felices",[
          ("Escarabajos Machacados",52,[f1,f2]),
          ("Ojo de Tigre Sucio",2,[f3])]),
     ("Multijugos",[
          ("Cuerno de Bicornio en Polvo",10,    [invertir3, (\(a,b,c) → (a,a,c))]),
          ("Sanguijuela hormonal",54,[(aplicar3 (*2)), (\(a,b,c) → (a,a,c)) ])]),
     ("Flores de Bach",[("Orquidea Salvaje",8,[f3]), ("Rosita",1,[f1])])
     ]
```

Donde cada tupla poción de la lista tiene la forma:
("Nombre de la Pocion", [("NombreIngrediente", cantidadEnGramos, [efecto])])

Los **efectos** son funciones que reciben una 3-upla de niveles (nivelSuerte, nivelPoderDeConvencimiento, nivelFuerzaFisica) y devuelven otra 3-upla con alguno o todos los niveles cambiados.

A partir de la base de conocimiento presentada, se pide resolver los siguientes puntos utilizando los conceptos aprendidos del paradigma funcional: composición, aplicación parcial, orden superior, listas por comprensión. Indicar en cada caso qué concepto se utilizó.
Tener en cuenta que recursividad solo puede utilizarse una única vez en el parcial, y el uso de listas por comprensión debe ser acotado, ya que siempre puede reemplazarse por funciones de orden superior.

1) Dada una tupla de niveles definir las funciones:

a) **sumaNiveles**, que suma todos los niveles
b) **diferenciaNiveles**, es la diferencia entre el nivel más alto y el nivel más bajo

Dada una tupla persona definir las funciones

c) **sumaNivelesPersona**, por ejemplo la suma de niveles de Harry es 20 (11+5+4).

```
> sumaNivelesPersona ("Harry",(11, 5, 4))
20
```

d) **diferenciaNivelesPersona**, que aplicada a Harry debería ser 7 (11 - 4).

```
> diferenciaNivelesPersona ("Harry",(11, 5, 4))
7
```

2) Definir la función **efectosDePocion**, que recibe una tupla poción y devuelve una lista con todos los efectos de cada uno de sus ingredientes.

```
> efectosDePocion ("Felix Felices",[("Escarabajos Machacados",5,[f1,f2]),("Ojo de Tigre",2,[f3])])
[f1,f2,f3]

--En Haskell las funciones no se pueden mostrar, esto es solo una consulta a modo de ej.
```

3) Definir la función **pocionesHeavies**, que recibe una lista de pociones y devuelve los nombres de las pociones que tienen al menos 4 efectos.

```
> pocionesHeavies misPociones
["Multijugos"]
```

4)

a) definir la funcion **incluyeA** que espera dos listas, devolviendo True si la primera está incluida en la segunda. P.ej.

```
incluyeA [3,6,9] [1..10]
```

devuelve `True`

b) definir la función **esPocionMagica**/1. Una poción es mágica si el nombre de alguno de sus ingredientes tiene todas las vocales y además de todos los ingredientes se pide una cantidad par.

En el ejemplo “Multijugos” es mágica, “Felix Felices” no porque ningún nombre incluye todas las vocales, “Flores de Bach” no porque hay un ingrediente con cantidad impar.

Hint: para lo de las vocales puede convenir usar el 4.a teniendo en cuenta que los Strings son listas y que es fácil definir una lista con las vocales.

5) Definir la función **tomarPocion,** que recibe una poción y una persona, y devuelve una tupla persona que muestra cómo quedaría la persona después de haber tomado la poción.

Cuando una persona se toma una poción, se le aplican todos los efectos de cada ingrediente de la poción, en orden.

Siendo

```
pocionFelix =
("Felix Felices",[("Escarabajos Machacados", 52, [f1,f2]),("Ojo de tigre sucio",2,[f3])])

> tomarPocion pocionFelix ("Harry",(11, 5, 4))
("Harry", (12, 7, 12))

--Porque le aplica f1, f2 y por último f3
--Pista: usar fold o recursividad
```

6) Definir la función **esAntidoto**, que recibe una persona y dos pociones, y devuelve true en caso de que la segunda poción revierta el efecto de la primera sobre la persona.

Es decir, si la persona queda igual después de tomar primero la primer poción y después la segunda.

7)

Definir la función **personaMasAfectada**, que recibe una poción, una ponderacion de niveles y una lista de personas, y devuelve la persona que después de tomarse la poción hace **máximo** el valor de la ponderación de niveles.

Una ponderación de niveles es una función que espera una terna de niveles (suerte,convencimiento,fuerza física) y devuelve un número.

No se puede usar recursividad, listas por comprensión, foldl ni definiciones locales.

8)

Escribir consultas que, usando la función del punto anterior, respondan la persona que quedó más afectada según las siguientes ponderaciones

a) suma de niveles (suerte, poder de convencimiento y fuerza física)
b) promedio de niveles (puede ser el promedio entero)
c) fuerza física
d) diferencia de niveles