Trabajo Práctico 1
Programación Funcional

Paradigmas ~~de Lenguajes~~ de Programación
1er cuatrimestre 2024

Fecha de entrega: 19 de abril

Cuando se creó el universo, no había nada, luego el Big Bang lanzó 6 cristales elementales a lo largo del universo virgen, cada una de estas gemas del infinito controla un aspecto básico de la existencia.

Introducción

Thanos ya viene y está juntando las gemas del infinito. Si las consigue todas, tendrá el poder suficiente para hacer que todes recursen la materia. Es su responsabilidad evitarlo. Pero no se preocupen, tenemos la ayuda de Dr. Strange que nos pasó su algoritmo para explorar 14,000,605 universos, con el cual podemos buscar uno en el que le ganemos a Thanos. En cada universo existen **personajes** (como *Thanos*, *Iron man* o *Thor*) y **Objetos** (como el *Escudo del Capitán América* o la *Gema del Tiempo*). Algunos objetos están sueltos y otros están en posesión de algún personaje.

Representación

Los personajes y objetos se representan con los tipos `Personaje` y `Objeto` respectivamente. Los personajes y objetos se inicializan con una posición (representada con el tipo `Posición` a partir de una tupla de `Float`s) y un nombre (un `String`). Los pesonajes se pueden mover en una de las cuatro direcciones (representadas con el tipo `Dirección`) y pueden morir. Cuando un personaje se mueve en una dirección una de las componentes de su posición se incrementa o decrementa en uno (cuál componente y si incrementa o decrementa depende de la dirección de movimiento). Los objetos pueden ser tomados por un personaje y pueden ser destruidos. Una vez que un objeto es tomado su posición pasa a ser la del personaje que lo tomó. Los universos se representan con listas cuyos elementos pueden ser personajes u objetos así que se utilizará el tipo `[Either Personaje Objeto]` para representarlos.

Ejercicios

Ejercicio 1

Definir el esquema de recursión estructural para los tipos `Personaje` (`foldPersonaje`) y `Objeto` (`foldObjeto`). Por tratarse de esquemas de recursión está permitido definirlos usando recursión explícita.

Ejercicio 2

Definir las funciones `posición_personaje` (que dado un personaje devuelve su posición actual) y `nombre_objeto` (que dado un objeto devuelve su nombre). Notar que está disponible la función `siguiente_posición :: Posición -> Dirección -> Posición` que dada una posición y una dirección devuelve la posición siguiente a la dada en la dirección dada.

Ejercicio 3

Definir las funciones `objetos_en` y `personajes_en` que dado un universo devuelven la lista de objetos y la lista de personajes de dicho universo, respectivamente. Luego demostrar la siguiente propiedad:
$\forall$ `u :: Universo` . $\forall$ `o :: Objeto` . `elem o (objetos_en u)` $\Rightarrow$ `elem (Right o) u`

Ejercicio 4

Definir la función `objetos_en_posesión_de` que dado un nombre de personaje y un universo, devuelve la lista de los objetos del universo que están en posesión del personaje con ese nombre.

Ejercicio 5

Definir la función `objeto_libre_mas_cercano` que dado un personaje y un universo devuelve el objeto del universo más cercano al personaje. Se puede asumir que hay al menos un objeto en el universo. En caso de empate se puede devolver cualquiera de ellos.

Ejercicio 6

Definir la función `tiene_thanos_todas_las_gemas` que dado un universo devuelve si *Thanos* (es decir, si algún personaje del universo con nombre "Thanos") ya tiene las 6 gemas en su posesión en ese universo (es decir, si la cantidad de objetos en su posesión que son gemas es igual a 6).

Ejercicio 7

Definir la función `podemos_ganarle_a_thanos` que toma un universo y devuelve si en ese universo tenemos alguna chance de ganarle a *Thanos*. Para que exista esa chance necesitamos en primer lugar que no haya pasado ya que *Thanos* haya conseguido todas las gemas (en ese caso ya perdimos) y por otro que *Thor* pueda vencerlo con *Stormbreaker* (es decir, que estén en el universo el personaje de nombre "Thor" y el objeto de nombre "StormBreaker") o que *Wanda* pueda destruir la *Gema de la Mente* matando a *Visión* (es decir, que estén en el universo los personajes de nombre "Wanda" y "Visión" y además que el personaje de nombre "Visión" esté en posesión del objeto de nombre "Gema de la Mente").

Pautas de Entrega

Se debe entregar a través del campus un único archivo llamado "tp1.hs" conteniendo el código con la implementación de las funciones pedidas. Para eso, ya se encuetra disponible la entrega "TP1 - Programación FuncionalTarea" en la solapa "Talleres" (configurada de forma grupal para que sólo una persona haga la entrega en nombre del grupo). El código entregado **debe** incluir tests que permitan probar las funciones definidas. El código debe poder ser ejecutado en Haskell2010. No es necesario entregar un informe sobre el trabajo, alcanza con que el código esté **adecuadamente** comentado (son comentarios adecuados los que ayudan a entender lo que no es evidente o explican decisiones tomadas; no son adecuadas las traducciones al castellano del código). Los objetivos a evaluar son:

- Corrección.

- Declaratividad.
- Prolijidad: evitar repetir código innecesariamente y usar adecuadamente las funciones previamente definidas (tener en cuenta tanto las funciones definidas en el enunciado como las definidas por ustedes mismos).
- Uso adecuado de funciones de alto orden, currificación y esquemas de recursión: Es necesario para los ejercicios que usen las funciones que vimos en clase y aprovecharlas, por ejemplo, usar zip, map, filter, take, takeWhile, dropWhile, foldr, foldl, listas por comprensión, etc, cuando sea necesario y no volver a implementarlas.

Salvo donde se indique lo contrario, **no se permite utilizar recursión explícita**, dado que la idea del TP es aprender a aprovechar las características enumeradas en el ítem anterior. Se permite utilizar listas por comprensión y esquemas de recursión definidos en el preludio de Haskell y los módulos `Prelude`, `List`, `Maybe`, `Data.Char`, `Data.Function`, `Data.List`, `Data.Maybe`, `Data.Ord` y `Data.Tuple`. Las sugerencias de los ejercicios pueden ayudar, pero no es obligatorio seguirlas. Pueden escribirse todas las funciones auxiliares que se requieran, pero estas no pueden usar recursión explícita (ni mutua, ni simulada con `fix`).

Importante: se admitirá un único envío, sin excepción alguna. Por favor planifiquen el trabajo para llegar a tiempo con la entrega.

Tests: cada ejercicio debe contar con uno o más ejemplos que muestren que exhibe la funcionalidad solicitada. Para esto se recomienda la codificación de tests usando el paquete HUnit https://hackage.haskell.org/package/HUnit. El esqueleto provisto incluye algunos ejemplos de cómo utilizarlo para definir casos de test para cada ejercicio.

Para instalar HUnit usar: `> cabal install --lib HUnit`

Para instalar cabal ver: https://wiki.haskell.org/Cabal-Install

Para ejecutar los tests usar: `> main` para todos los tests, `> do runTestTT testsEjN` para los tests del ejercicio N.

Referencias del lenguaje Haskell

Como principales referencias del lenguaje de programación Haskell, mencionaremos:

- **The Haskell 2010 Language Report:** el reporte oficial de la última versión del lenguaje Haskell a la fecha, disponible online en http://www.haskell.org/onlinereport/haskell2010.
- **Learn You a Haskell for Great Good!**: libro accesible, para todas las edades, cubriendo todos los aspectos del lenguaje, notoriamente ilustrado, disponible online en http://learnyouahaskell.com/chapters.
- **Real World Haskell**: libro apuntado a zanjar la brecha de aplicación de Haskell, enfocándose principalmente en la utilización de estructuras de datos funcionales en la "vida real", disponible online en http://book.realworldhaskell.org/read.
- **Hoogle**: buscador que acepta tanto nombres de funciones y módulos, como signaturas y tipos *parciales*, online en http://www.haskell.org/hoogle.
- **Hayoo!**: buscador de módulos no estándar (i.e. aquéllos no necesariamente incluídos con la plataforma Haskell, sino a través de **Hackage**), online en http://holumbus.fh-wedel.de/hayoo/hayoo.html.