Trabajo Práctico 1
Programación Funcional

Paradigmas de Lenguajes de Programación — 2^{do} cuat. 2021

Fecha de entrega: 9 de Septiembre de 2021

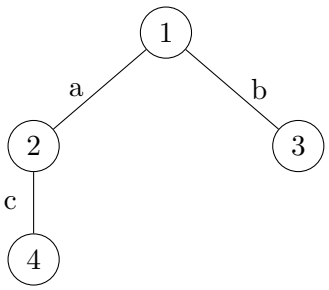
1. Introducción

Este trabajo práctico consiste en implementar en Haskell y luego testear varias funciones que modelan **Rosetrees etiquetados** de la siguiente manera:

```
data RTE a = Rose a  [(Char,RTE a)]
```

Cada nodo del Rosetree tendrá un valor de tipo `a` y una lista de hijos a los cuales se puede acceder a través de una etiqueta de tipo `Char`.

Por ejemplo, `unRose = Rose 1 [('a', Rose 2  [('c', Rose 4  [])]),('b', Rose 3  [])]` es el Rosetree:



2. Resolver

Ejercicio 1

Definir la igualdad para el tipo `RTE a` de manera que este tipo sea una instancia de `Eq`. Un nodo `Rose` es igual a otro si tienen el mismo valor y los mismos hijos (aunque estén en otro orden). En este punto se permite usar recursión explícita **solo en la estructura del Rosetree**.

Ejercicio 2

a) Definir y dar el tipo de la función `foldRTE` que implementa el esquema de recursión estructural sobre Rosetrees. En este punto se permite usar recursión explícita **solo en la estructura del Rosetree**.

b) Definir y dar el tipo de la función mapRTE que recibe una función que se aplica a todos los valores de los nodos de un Rosetree.

Ejercicio 3

a) Definir la función nodos :: RTE a → [a] que dado un Rosetree, devuelve una lista con los valores de cada nodo en algún orden. Por ejemplo, nodos unRose = [1,2,3,4].

b) Definir la función altura :: RTE a → Int que dado un Rosetree devuelve su altura. Por ejemplo, altura unRose = 3.

Ejercicio 4

Definir la función etiquetas :: RTE a → [Char] que dado un Rosetree, devuelve una lista con todas sus etiquetas en algún orden. Por ejemplo, etiquetas unRose = ['a','b','c'].

Ejercicio 5

Definir la función ramas :: RTE a → [String] que dado un Rosetree, devuelve una lista con cada una de las ramas que forman las etiquetas desde la base hasta las hojas. Por ejemplo,
ramas unRose = ["ac","b"].

Ejercicio 6

Definir la función subRose :: RTE a → Int → RTE a que dado un Rosetree y una altura, devuelve el Rosetree podado en ese nivel. Por ejemplo, subRose unRose 2 = Rose 1 [('a', Rose 2 []),('b', Rose 3 [])] .
Se puede asumir que la altura es mayor a 0.

Tests

Parte de la evaluación de este Trabajo Práctico es la realización de tests. Tanto HUnit[1] como HSpec[2] permiten hacerlo con facilidad.

En el archivo de esqueleto que proveemos se encuentran tests básicos utilizando *HUnit*. Para correrlos, ejecutar dentro de *ghci*:

```
> :l TP1.hs
[1 of 1] Compiling TP1              ( TP1.hs, interpreted )
Ok, one module loaded.
> tests
```

Para instalar HUnit usar: `> cabal install HUnit` o bien `apt install libghc-hunit-dev`.

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

[1] https://hackage.haskell.org/package/HUnit
[2] https://hackage.haskell.org/package/hspec

Pautas de Entrega

Todo el código producido por ustedes debe estar en el archivo `TP1.hs` y es el **único** archivo que deben entregar. Cada función (incluso las auxiliares) asociada a los ejercicios debe contar con un conjunto de casos de test que muestren que exhibe la funcionalidad esperada. Para esto se deben agregar casos de test en las funciones *testEjercicio* en el archivo fuente *taller.hs*, agregando todos los tests que consideren necesarios (se incluyen algunos tests de ejemplo). Se debe enviar un e-mail a la dirección plp-docentes@dc.uba.ar. Dicho mail debe cumplir con el siguiente formato:

- El título debe ser `[PLP;TP-PF]` seguido inmediatamente del **nombre del grupo**.
- El código Haskell debe acompañar el e-mail y lo debe hacer en forma de archivo adjunto con nombre `TP1.hs` (puede adjuntarse un `.zip` o `.tar.gz`).
- 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 lo ya definido. Se permite utilizar listas por comprensión y esquemas de recursión definidos en el preludio de Haskell y los módulos `Prelude`, `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.

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. aquellos no necesariamente incluídos con la plataforma Haskell, sino a través de **Hackage**), online en http://holumbus.fh-wedel.de/hayoo/hayoo.html.