Proyecto 1

Algoritmos y Estructuras de Datos I
Laboratorio

23 de agosto de 2010

El objetivo de este práctico es recordar (o aprender) cómo definir funciones en Haskell. En particular, se evaluará la definición de funciones usando caso base y caso inductivo (a través de análisis por casos, o pattern-matching). Otra de las cosas que se evaluarán es el uso de las funciones `foldr` y `map` , que se pueden usar para definir muchas funciones polimórficas sobre listas. En algunas de las funciones será necesario utilizar definiciones locales y también el uso de guardas para alternativas booleanas. En otros casos deberás utilizar lo que en Haskell se llaman "sections", que corresponde a la aplicación parcial de operadores binarios.

En todos los ejercicios escribe el tipo (y la clase) al que pertenece la función. Cuando haya que definir dos versiones de la función `ejemplo`, llama a la segunda versión `ejemplo'`.

1. Hacer una función `minimo` que devuelva el mínimo de una lista.

   a) Definirla usando caso base y caso inductivo.

   b) Definirla utilizando el funcional `foldr` ; en este caso debemos utilizar una función del preludio de Haskell que devuelva el mínimo entre dos expresiones comparables.

   Ayuda.

   - ¿Qué clase se debe utilizar para definir el tipo de la función?
   - Como deseamos que la función sea total, debemos tener cuidado con el caso base. Investiga a qué otra clase debería pertenecer el tipo de los elementos de la lista para poder hacerlo.

2. Considera las siguientes funciones:

   - `sumarALista :: Num a => a -> [a] -> [a]` que toma un número y una lista de números y le suma a cada elemento de la lista el primer parámetro. Por ejemplo:

     ```
     sumarALista 3 [4,6,7] = [7,9,10]
     ```

   - `encabezar :: a -> [[a]] -> [[a]]` que toma una expresión de tipo `a` y lo pone en la cabeza de cada lista del segundo parámetro. Por ejemplo:

     ```
     encabezar 3 [[2,1],[],[4,7]] = [[3,2,1],[3],[3,4,7]]
     ```

   a) Escribe `sumarALista` y `encabezar` usando caso base y caso inductivo.

   b) Escribe ambas funciones utilizando el funcional `map` . Esta vez, no puedes usar más de una línea, para cada una; y tampoco puede haber definiciones locales.

3. Considera la función `encuentra` que dado un valor de tipo `Int` y una lista de pares `[(Int,String)]` devuelve el segundo componente del primer par cuyo primer componente es igual al primer parámetro. En el caso que ningún elemento de la lista cumpla con esto, devolver el string vacío. Por ejemplo:

   ```
   encuentra 10 [(40,"tos"),(10,"uno"),(16,"taza"),(10,"dos")] = "uno"
   encuentra 102 [(40,"tos"),(103,"vela"),(16,"taza")] = ""
   encuentra 102 [] = ""
   ```

a) Define la función `encuentra`, sin usar `foldr` .

b) Define esta función utilizando el funcional `foldr` .

c) (Punto estrella) ¿Se puede generalizar el tipo de esta función?

Ayudas.

- Recuerda que en Haskell `String` es lo mismo que lista de `Char`; esto es un recordatorio, no significa que precises este dato en tus definiciones.
- Pensar inducción sobre la lista para el primer punto.
- Para el punto estrella, piensa cuál de los tipos se puede generalizar sin problemas y cuál puede ser problemático.

4. - La función `primIgualesA` toma un valor y una lista y devuelve el tramo inicial más largo de la lista cuyos elementos son iguales al valor. Por ejemplo:

```
primIgualesA 3 [3,3,4,1] = [3,3]
primIgualesA 3 [4,3,3,4,1] = []
primIgualesA 3 [] = []
primIgualesA 'a' "aaadaa" = "aaa"
```

   - La función `primIguales` toma una lista y devuelve el mayor tramo inicial de la lista cuyos elementos son todos iguales entre sí. Por ejemplo:

```
primIguales [3,3,4,1] = [3,3]
primIguales [4,3,3,4,1] = [4]
primIguales [] = []
primIguales "aaadaa" = "aaa"
```

a) Programar `primIgualesA` con caso base e inductivo.

b) Programar `primIgualesA` usando `foldr` .

c) En el preludio existe una función que es más general que `primIgualesA`; utilizá esa función para programar `primIgualesA` en una sóla linea.

d) Programar con caso base e inductivo `primIguales`.

e) Usar cualquier versión de `primIgualesA` para programar `primIguales`.

f) ¿Se puede definir `primIguales` sin hacer inducción?

Ayuda y más preguntas.

- Puedes consultar la sección 11.5 del libro sobre el concepto de generalización.
- `foldr` es una manera de hacer inducción sobre listas. ¿Por qué?
- ¿Podrías describir una forma de transformar una función definida usando caso base y caso inductivo en una función usando sólamente `foldr` ?

5. Derivar los ejercicios 3b), 3c), 3d) y 3e) del práctico 2 de la materia y **después** implementar las funciones obtenidas en el formalismo básico en Haskell.

Muy importante! No escriba una línea de código Haskell hasta que no termine de derivar la función en lapiz y papel (le recomendamos que mientras deriva apague el monitor de la computadora).

Para aprobar este ejercicio además hay que tener en cuenta:

- Al momento de la evaluación hay que presentar en papel las funciones en el formalismo básico y sus derivaciones.
- Las funciones resultados de las derivaciones deben corresponder a las implementadas en Haskell. Es decir, deben tener los mismos nombres, argumentos, estilo pattern matching o por casos, etc.
- Al momento de la evaluación debe dar una explicación sobre el comportamiento operacional de las funciones. Esto es, debe poder explicar que hacen las funciones escritas en Haskell, de la misma manera que en los ejercicios anteriores.