Proyecto Lenguaje de Programación: Parser XML en Haskell

Gabriel Falcones Paredes

January 19, 2014

1 Objetivos

- Implementar un Parser de archivos XML utilizando el Lenguaje de Programación Haskell.
- Usando el parser XML desarrollado, utilizarlo para interpretar el archivo wurfl.xml proporcionado por el profesor.
- Desarrollar metodos para realizar consultas en el archivo wurfl.xml

2 Introducción

El siguiente proyecto consiste en la implementación de un parser para archivos tipo XML, desarrollado en el Lenguaje de Programación Haskell, y utilizarlo para interpretar el archivo wurlf.xml. Una vez conseguido interpretar este archivo, ser debera ser capaz de realizar consultas acerca de los contenidos de tal archivo.

3 Alcance del proyecto

- Lectura de archivos xml.
- Desarrollo de Estructura de Datos para almacenar los contenidos de un archivo xml.
- Parseo del archivo xml y almacenamiento de su contenido en la estructura de datos.
- Implementación de funciones de consulta de los contenidos de la estructura de datos.

3.1 Lectura de archivos xml.

- Se desarrollo la funcion readXML que recibe el nombre de un archivo xml y devuelve su contenido en un IO String.
- El IO String a su vez es transformado a un String para almacenarlo en una estructura de datos.

```
-- Lee Archivo .xml
readXML :: String -> IO String
readXML file = do
  handle <- openFile file ReadMode
  hGetContents handle
```

3.2 Desarrollo de la Estructura de datos

- Se creo una nueva estructura llamada XMLTree, la cual esta formada por un String que representa el nombre del tag, una lista de atributos, y una lista de nodos hijos que a su vez también son elementos tipo XMLTree.

```
data XMLTree = Empty |  XMLTree { tag :: String
                    , attr :: [(String,String)]
                    , childs :: [XMLTree] } deriving (Show)
```

3.3 Parseo del archivo xml y almacenamiento de su contenido en la estructura de datos

- A partir del String creado a partir del archivo xml se crea una lista formada por cada una de las palabras del string.
- Cada elemento de esta lista se analiza para comprobar si es un tag, atributo, etc; y de acuerdo a esto se va construyendo la estructura del XMLTree.

```
crearXMLTreeAux :: XMLTree -> [String] -> (XMLTree,[String])
crearXMLTreeAux tree [] = (tree,[])
crearXMLTreeAux tree (x:xs)
  | isCloseAttrTag x = ((addAttr tree x),xs)
  | isAttr x = crearXMLTreeAux (addAttr tree x) xs
  | isCloseTagName x = (tree,xs)
  | isTagName x = case tree of Empty -> crearXMLTreeAux (newXMLTree (crearTagName x)) xs
                               _ -> let newChildTuple =
                                          crearXMLTreeAux (newXMLTree (crearTagName x)) xs
                                    in let newTree =
                                             XMLTree (tag tree) (attr tree) (childs tree ++
                                                                             [fst newChildTuple])
                                       in crearXMLTreeAux newTree (snd newChildTuple)
  | otherwise = crearXMLTreeAux tree (unirAttr (x:xs))
```

3.4 Implementación de funciones de consulta

- Utilizando la estructura XMLTree creada, se implementaron funciones para consultar el contenido de la misma, de acuerdo a los requisitos pedidos por el profesor durante la sustentación.

```
buscarNumFallback :: [String] -> String -> [String] -> [String]
buscarBuiltCamera :: [String] -> String -> [String] -> [String]
buscarNumDevices :: [String] -> Integer
buscarInId :: [String] -> [String] -> String
```

4 Fuera del alcance del proyecto

- El proyecyo realizado fue diseñado exclusivamente para el archivo wurfl.xml proporcionado.
- Un documento xml puede incluir contenido dentro de sus tags aparte de los tags hijos. Este caso no se implemetó debido a que en el archivo wurfl.xml no existian tags con contenido.
- Además no se considera el caso de que exista espacios entre los nombres de los tags.

5 Conclusiones

- Se logró concluir el proyecto, guardando correctamente el archivo xml en la estructura creada.
- Para el desarrollo en Haskell fue necesario aprender el paradigma de los lenguajes funcionales, ya que es completamente diferente a los lenguajes imperativos, pero genera metodos de resolución más elegantes.
- Fue necesario aprender a desarrollar la estructura con XML, aunque es relativamente sencillo.
- Hubieron complicaciones durante la lectura de archivos en Haskell, ya que fue resulto un poco confuso la manera en que Haskell soluciona la lectura de archivos, ya que conlleva la utilizacion de metodos impuros.
- Al momento de realizar las consultas y de mostrar el contenido del árbolXML, se utilizó un archivo xml de prueba para obtener resultados rápidos. Debido al metodo de evaluación de Haskell (lazy evaluation), el obtener los resultados de consultas para el archivo wurfl.xml tomaba una enorme cantidad de tiempo, ya que debía de leer un archivo muy pesado, guardarlo en la estructura, y luego leer

la estructura. Debió de tomarse en cuenta los temas de eficiencia para desarrollar una solución más eficiente.