Pràctica obligatòria de Haskell. XML Tree search (Completa)

1 Presentació

Els documents XML són similars als HTML, però les etiquetes (tags) són definides per l'usuari en lloc de ser etiquetes fixes HTML. Els documents XML es poden veure com a arbres. Considerem el següent document XML:

```
<llibres>
  <llibre any="2004" edicio="1">
    <titol>Razonando con Haskell</titol>
    <autor>Blas C. Ruiz</autor>
    <autor>Francisco Gutierrez</autor>
    <autor>Pablo Guerrero</autor>
    <autor>Jose E. Gallardo</autor>
  </llibre>
  <llibre edicio="2" any="1999">
    <titol>HASKELL: The Craft of Functional Programming</titol>
    <editor>A. D. McGettrick</editor>
    <autor>Simon Thompson</autor>
  </llibre>
  <llibre edicio="1" any="2000">
    <titol>Programming language pragmatics</titol>
    <autor>Michael L. Scott</autor>
  </llibre>
</llibres>
```

Té una única arrel que està delimitada per l'*etiqueta* (tag) d'inici `<llibres>` i l'etiqueta de final `</llibres>`. Aquesta arrel té tres elements (fills) de tipus `llibre`. Cada element de tipus `llibre` té un *atribut* de tipus `any` i un de tipus `edicio` (que són terminals), així com un element de tipus `titol` i, possiblement, d'altres de tipus `autor` i `editor` com a fills. Cada element `autor`, `editor` i `titol` té un únic fill de tipus Text, per tant aquests també són terminals. Els valors dels atribut s'escriuen entre dobles cometes després del símbol de igualtat (`=`). En canvi, els de tipus Text s'escriuen sense cometes.

Per tant, en els arbres XML que considerem aquí, podem considerar que hi ha tres tipus de nodes (en els reals n'hi ha set): element, attribute, text. Els nodes element contenen el tag, els nodes atribut contenen un atribut i el seu valor que és un string, i els text inclouen informació textual (un string). Els nodes atribut i text són terminals.

2 Primera part. Els XMLTree.

1. Definiu en Haskell el tipus de dades `XMLTree` que s'ha explicat abans com un Tree de XMLNode on Tree és un data genèric i XMLNode és un data que admet els tres tipus de node esmentats.

- definiu correctament la funció de mostrar en el tipus `XMLTree` com a instància de la classe `Show`, de manera que el resultat sigui un String en format XML, és a dir, amb els tags corresponents i els espais en blanc i salts de línia tal que si s'escriu amb un `putStr` es mostra exactament com s'ha presentat en el primer exemple.

  Important: per a poder fer un instance del XMLTree, que no és un data, cal cridar al ghci amb el flag -XTypeSynonymInstances (si `GHCI<=7.0.4`) o -XFlexibleInstances (si `GHCI>=7.2.1`), és a dir

  ```
  ghci -XTypeSynonymInstances practica1.hs # Si GHCI<=7.0.4
  ghci -XFlexibleInstances practica1.hs # Si GHCI>=7.2.1
  ```

  Alternativament, podeu aconseguir el mateix efecte posant al principi del fitxer de la pràctica la línia:

  ```
  {-# LANGUAGE TypeSynonymInstances #-}
  ```

  o, corresponentment,

  ```
  {-# LANGUAGE FlexibleInstances #-}
  ```

- definiu correctament la funció de llegir en el tipus `XMLTree` que es digui `readXMLTree` amb el següent tipus:

  ```
  readXMLTree :: String -> XMLTree
  ```

  És a dir, a partir d'un String que no contindrà blancs o salts de línia innecessaris (a banda dels inicials o quan acaba un node element o text) i que segueix la sintaxi d'un arbre XML, retorna un `XMLTree`.

  Feu també les funcions:

  ```
  readElementNode :: String -> XMLNode
  readAtributeNode :: String -> XMLNode
  readTextNode :: String -> XMLNode
  ```

  La primera rep un String amb l'etiqueta, la segona un String amb la igualtat i la tercera un String amb el text.

  NO heu de fer que XMLTree sigui instància de la classe `Read`

Podeu definir constants al vostre programa per tal de no escriure cada vegada l'`XMLTree` amb que treballeu:

```
llibresArbre :: XMLTree
llibresArbre = ...
```

També podeu definir (opcionalment) renombrats de tipus per millorar la llegibilitat.

3 Segona Part. Definició de consultes d'arbre.

Volem poder fer consultes sobre els XMLTree, però definint un llenguatge de "queries", que barreja part del llenguatge XPath amb part del llenguatge de les expressions regulars. Per això seguirem les següents passes.

3.1 La Class `QNode`

Pretenem treballar de forma genèrica per a que el nostre codi es pugui adaptar a arbres construïts amb diferents tipus de nodes. És a dir, que si decidim estendre o modificar el nostre XMLTree tenint altres tipus de nodes, bona part de la feina es podrà reaprofitar. Per això heu de definir

1. la class `QNode` tal que un tipus `a` és de la classe si té tres operacions que reben dos valors de tipus `a` i retornen un booleà:

   (a) `isKind`: que ens indica si el primer objecte és del mateix tipus del segon.

   (b) `geq`: que ens indica si el primer objecte és (en algun sentit) més gran o igual que el segon.

   (c) `leq`: que ens indica si el primer objecte és (en algun sentit) més petit o igual que el segon.

2. Feu que `XMLNode`, sigui instance de la classe `QNode`. Per això, en el isKind considereu que són del mateix tipus si tenen el mateix constructor i

   (a) si és "element" o "text", o bé contenen el mateix `String` o el del segon node és buit.

   (b) si és "attribute", o bé contenen els mateixos dos `String`, o bé contenen el mateix primer `String` i el segon del segon objecte és buit, o bé els dos del segon objecte són buits.

   L'operació `geq`, només és certa si els dos són nodes "attribute" tenen el primer `String` igual i el segon `String` del primer convertit en `Int` és més gran o igual que el segon `String` del segon convertit en `Int`. Podeu assumir que sempre es podran convertir en `Int`, si apareixen en un `geq`.

   L'operació `leq`, és igual que la `geq` però amb menor o igual.

3.2 El data `Condition`

1. Primer heu de crear un nou data polimòrfic `Condition n`, que admet com a constructors:

   (a) `IsKind`, que te un paràmetre de tipus `n` i que representa la condició de ser de la mateixa forma del paràmetre.

(b) `Geq` i `Leq` que reben un valor de tipus `n` i que representen, respectivament, la condició de ser més gran o igual (o més petit o igual) que el paràmetre.

(c) `CTrue` i `CFalse` que representen respectivament les condicions certa i falsa, i `CNot`, `CAnd` i `COr` que representen respectivament la negació, la conjunció i la disjunció de condicions.

2. Definiu l'operació genèrica `evaluate` que rep un paràmetre de tipus `a` i un de tipus `(Condition a)` i retorna cert si el primer paràmetre satisfà la condició del segon paràmetre i fals en altre cas. Noteu que cal que `a` sigui de la classe `QNode`.

3.3 El tipus `QTree` i les consultes

Volem realitzar consultes sobre els arbres genèrics, que ens permetin seleccionar nodes o subarbres, segons l'ús que en volem fer. Per això seguiu les següents passes:

1. Definiu un data `Range` que permet selecciona un prefix, un sufix o un element determinat, amb els següents constructors:

   (a) `PGe` que té un `Int` com a paràmetre i indica totes les posicions més grans o iguals que l'enter.

   (b) `PLe` que té un `Int` com a paràmetre i indica totes les posicions més petites o iguals que l'enter.

   (c) `PEq` que té un `Int` com a paràmetre i indica la posició de l'enter.

2. Definiu un data polimòrfic `QTree` que representa una consulta (que indica quins subarbres es pot arribar) en un arbre amb els següents constructors:

   (a) `ThisTree` i que permet seleccionar l'arbre en curs.

   (b) `AnyTree` que permet seleccionar l'arbre en curs i tots els seus subarbres.

   (c) `Selection` que té un paràmetre de tipus `Range` i un de tipus `QTree` i que representa un selecció del resultat de la query del segon paràmetre.

   (d) `RNode` que té un paràmetre de tipus `Condition` i un de tipus llista de `QTree` i que representa la consulta que demana la condició a l'arrel i la llista de consultes als fills (una a una). Si la llista de consultes als fill és buida, vol dir que el subarbre en curs satisfà la consulta. Si n'hi ha menys consultes que que fills, es considera per defecte que les que no hi són són `ThisTree` i si en sobren, no s'apliquen.

   (e) `StarTree` que té un paràmetre de tipus `Condition` i un de tipus `QTree` i que representa que la condició és pot aplicar zero o més vegades. Per tant, la consulta (segon paràmetre) sempre s'aplica a l'arbre en curs i si la condició es compleix per l'arrel, llavors (també) es torna a aplicar la mateixa `StarTree` consulta als fills.

(f) `PlusTree` és igual que el `StarTree`, però és obligatori que la condició (primer paràmetre) es compleixi per l'arrel. Si no és el cas ningú satisfà la consulta, en altre cas és com el `StarTree`.

(g) `Union` que rep un a llista de `QTree` i representa la unió d'una llista de consultes.

3. Feu una operació genèrica `xQueryTree` que té com a paràmetre un `(QTree a)` i un `(Tree a)` i retorna la llista de tots els subarbres que satisfan la consulta. Noteu que caldrà que el tipus `a` sigui de la classe `QNode`. Per exemple, amb la consulta

   ```
   StarTree CTrue (RNode (IsKind (readElementNode "llibre")) [])
   ```

   sobre el XMLTree de l'exemple inicial ens ha de tornar la llista que conté els tres fills XMLTree que tenen com a arrel un node amb tag "llibre".

4. Feu una operació genèrica `xQueryNode` que té com a paràmetre un `(QTree a)` i un `(Tree a)` i retorna la llista de tots els nodes arrel dels subarbres que satisfan la consulta. Noteu que caldrà que el tipus `a` sigui de la classe `QNode`.

5. Feu una operació genèrica `xRelative` que té com a paràmetre dos `(QTree a)` i un `(Tree a)`, i retorna la llista de tots els subarbres que satisfan la primera consulta i que tenen algun subarbre que satisfà la segona. Per exemple, amb les consultes

   ```
   StarTree CTrue (RNode (IsKind (readElementNode "llibre")) [])
   ```
   i
   ```
   StarTree CTrue (RNode (CAnd (Geq (readAtributeNode "any=1999"))
   (Leq (readAtributeNode "any=2001"))) [])
   ```

   sobre el XMLTree de l'exemple inicial ens ha de tornar la llista que conté els dos fills XMLTree que tenen com a arrel un node amb tag "llibre" i algun atribut any entre 1999 i 2001.