Pràctica obligatòria de Haskell. XML search

1 Presentació

Els documents XML són similars als HTML, però les etiquetes (tags) són definides per l'usuari enlloc 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* (a l'etiqueta d'inici) de tipus `any` i un de tipus `edicio` 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 són *fulles*. Els valors dels atribut s'escriuen entre dobles cometes després del símbol de igualtat (`=`). En canvi, els fill de tipus Text s'escriuen sense cometes.

Es podria formalitzar aquesta mena de documents en forma d'arbre mitjançant un tipus Haskell, que anomenarem **obligatòriament** `XMLTree`, tal que representi qualsevol element XML. Per això teniu en compte la següent definició dels element XML. Un element XML (de fet, un arbre XML) és

- o bé un Text (per exemple, Programming language pragmatics),
- o un node intern de l'arbre amb les següents característiques:
  - una *etiqueta* definint el tipus XML del node intern. Per exemple `llibres` o `editor`.
  - un *conjunt de parells* de la forma: *atribut=valor*. Com ara el primer llibre que té `any="2004" edicio="1"`.
    Fixeu-vos que l'ordre dels atributs no és rellevant i que cada atribut pot aparèixer com a molt un cop a l'etiqueta d'inici. També fixeu-vos que el conjunt pot ser buit.

– un *nombre indefinit de fills* que són elements XML (i per tant `XMLTree` serà un tipus recursiu). Per exemple, el primer element `llibre` té un element `titol` i quatre elements `autor` com a fills. Compte perquè l'ordre dels fills és rellevant (de fet, a l'exemple podem parlar de primer, segon, ... autor d'un llibre).

Tingueu en compte que això són unes directrius sobre com definir l'estructura. Podeu fer variacions però consulteu-les abans amb el professor.

2 Es demana

1. Definiu en Haskell el tipus de dades `XMLTree` que s'ha explicat abans.

   - 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 l'espaiat/tabulat i salts de línia tal que si s'escriu amb un `putStr` es mostra exactament com s'ha presentat en el primer exemple.

   - 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 pot contenir blancs i salts de línia (innecessaris) i que segueix la sintaxi d'un element XML, retorna un `XMLTree`.

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

   - definiu correctament la igualtat en el tipus `XMLTree` com a instància de la classe `Eq`. Fixeu-vos que la igualtat estructural no val perquè hi ha components que són conjunts i no pas llistes...

   - definiu correctament la funció d'ordre `compare` en el tipus `XMLTree` com a instància de la classe `Ord`. Dos `XMLTree` es comparen de la següent manera:

     1. Dos textos es comparen amb l'ordre de Strings.
     2. Un node intern és més gran que un text.
     3. Dos nodes interns es comparen primer amb els fills d'esquerra a dreta i en cas d'igualtat de tots els fills es comparen els tags (com a String) i si persisteix la igualtat el conjunt d'atributs seguint el següent criteri:
        (a) Primer per mida del conjunt.
        (b) Segon comparant (amb l'ordre de Haskell) les parelles `(Atribut,Valor)` més grans dels dos conjunt (considerant tant l'atribut com el valor com a String).
        (c) Si persisteix la igualtat continuem comparant les següents dues parelles més grans dels dos conjunts i, així, successivament.

Noteu que, si donats dos conjunts de parelles d'atributs i valors no podem dir que cap és més gran que l'altre, és que són iguals.

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 com ara:

```
type Tag = String
type Atrib = String
```

2. Feu la funció `tagged::String --> XMLTree -> [XMLTree]` tal que donat un Tag `t` i un XMLTree `a` ens retorni la llista ordenada dels XMLTree de `a` que tenen `t` a l'arrel. Per exemple, essent `llibresArbre` l'`XMLTree` corresponent al document XML de l'exemple, el resultat seria:

```
tagged "llibre" llibresArbre
[<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>
,<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>
]

tagged "autor" llibresArbre
[<autor>Blas C. Ruiz</autor>
,<autor>Francisco Gutierrez</autor>
,<autor>Jose E. Gallardo</autor>
,<autor>Michael L. Scott</autor>
,<autor>Pablo Guerrero</autor>
,<autor>Simon Thompson</autor>
]
```

3. Sigui `type Condicio = (Atribut, (String -> Bool))`.
Feu la funció de consulta:
`search::Tag -> Condicio -> [(Tag,String)] -> XMLTree-> [XMLTree]`
tal que `search t (a,c) [lc] x`, ens torna la llista dels `XMLTree` de `x` amb tag arrel `t` i que satisfacin la condició `c` en el seu atribut `a` i que té un subarbre amb tag `tt` i un Text `ss` a sota, per a un parell `(tt,ss)` de `lc`. Si no volem imposar cap condició als atributs, usarem la cadena buida, és a dir, `""`. Exemples:

```
search "llibre" ("any",\x->x=="1999") [] llibresArbre
[<llibre edicio="2" any="1999">
  <titol>HASKELL: The Craft of Functional Programming</titol>
  <editor>A. D. McGettrick</editor>
  <autor>Simon Thompson</autor>
</llibre>
]

search "llibre" ("edicio",\x->x>="1") [("titol","Program")] llibresArbre
[<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>
]

search "llibre" ("",\x->True) [("titol","Razonando"),("editor","Scott")] llibresArbre
[<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="1" any="2000">
  <titol>Programming language pragmatics</titol>
  <autor>Michael L. Scott</autor>
</llibre>
]

search "llibres" ("",\x->True) [("titol","Java")] llibresArbre
[]
```