NFP 119 : Programmation Fonctionnelle

Session du 6 septembre 2010 – Corrigé

Il vous est recommandé de rédiger de façon concise et précise, tout en écrivant lisiblement. Il sera tenu compte de la présentation dans la note de la copie. Tous les documents sont autorisés. Le barème n'est donné qu'à titre indicatif

Exercice 1 (6 points):

On veut modéliser, à l'aide des listes, l'indexation d'un ensemble de documents selon leurs mot-clés. Dans cet index, chaque document est représenté par un enregistrement $\{titre = t; motsCle = l\}$, où t est le titre du document, et l est sa liste de mots-clés.

```
type documentIndexe = {titre: string; motsCles: string list};;

type index = documentIndexe list;;

let i=[ {titre="Amon-Re"; motsCles= ["Egypte"; "dieux"]};
       {titre="Voyager␣en␣Egypte"; motsCles= ["Egypte"; "voyages"]};
       {titre="Typeful␣Programming"; motsCles= ["typage"; "programmation";"informatique"]};
       {titre="Java␣pour␣les␣nuls"; motsCles= ["programmation";"informatique";"Java"]}
     ];;
```

Afin d'exploiter ces index vous écrirez les fonctions suivantes :

1. La fonction `titresParMot` qui prend un index i et un mot-clé m, et qui renvoie la liste de tous les titres (et uniquement les titres) de documents dont les mots clés contiennent m. La fonction `List.mem` peut-être utile ici.
   Réponse (1 pt) :

```
(* Question 1 *)

 let rec titresParMot m indx = match indx
 with  []   -> []
 | {titre=t; motsCles= l}::r ->
     if List.mem m l then t::(titresParMot m r)
     else titresParMot m r;;

titresParMot "Egypte" i;;
```

2. La fonction `tousInclus` qui prend deux listes l_1 et l_2 et teste si tous les éléments de l_1 sont inclus dans l_2. Les fonctions `List.for_all` et `List.mem` peuvent être utiles ici.
   Réponse : (1 pt)

```
(* Question 2 *)

let estInclus l1 l2 = List.for_all (fun x -> List.mem x l2) l1;;
```

3. La fonction `titresParTousLesMots` qui prend un index i et une liste de mots clés m, et qui renvoie la liste de tous les titres (et uniquement les titres) de documents qui contiennent tous les mots de m. La fonction `tousInclus` de la question précédente pourra être employée ici.
   Réponse : (1 pt)

```
(* Question 3 *)

let rec titresParTousLesMots lm indx = match indx
 with  []   -> []
   | {titre=t; motsCles= l}::r ->
              if estInclus lm l then t::(titresParTousLesMots lm r)
               else titresParTousLesMots lm r;;

titresParTousLesMots ["programmation";"informatique"] i;;
```

4. La fonction `pertinenceDe` qui prend un document indexé $d = \{titre = t; motsCles = l\}$ et une liste de mots clés l_m et retourne le nombre d'éléments de l_m qui sont dans l, ce qui donne une mesure de la pertinence du document d par rapport à la liste de mots-clés l_m. **Exemples** :

```
# pertinenceDe {titre="Voyager␣en␣Egypte";
              motsCles=["Egypte"; "voyages"]} ["Egypte"; "dieux"]};;
 -: int = 1

# pertinenceDe {titre="Voyager␣en␣Egypte";
              motsCles=["Egypte"; "voyages"]} ["Egypte"; "voyages"]}
 -: int = 2
```

   Réponse : (1.5 pt)

```
(* Question 4 *)

let pertinenceDe {titre=t; motsCles= lt} lm =
 let rec compte lm =
 match lm
 with [] -> 0
   | m::r -> if List.mem m lt then 1+ (compte r)
             else compte r
 in compte lm;;

pertinenceDe {titre="Voyager␣en␣Egypte"; motsCles=["Egypte"; "voyages"]}
["Egypte"; "dieux"];;
```

5. Ecrivez la fonction `lePlusPertinent` qui prend une liste de mots-clés et un index et renvoie le titre le plus pertinent de l'index, à savoir, celui dont la liste de mots clés contient le plus de mots recherchés. *Indice* : on peut employer une fonction auxiliaire qui calcule pour chaque document d une paire (t, n) composé de son titre t et de sa pertinence n. Votre fonction devra renvoyer le titre t pour lequel n est le plus grand.
   Réponse : (1.5 pt)

```
let lePlusPertinent lm indx =
```

```
let rec maxDoc i =
match i
with [] -> failwith "lePlusPertinent"
 |  [{titre=t} as x] -> (t, pertinenceDe x lm)
 |  ({titre=t} as x)::r ->
             let (tr,maxr) = maxDoc r
             and n = pertinenceDe x lm in
             if n>maxr then (t,n) else (tr,maxr)
 in fst(maxDoc indx);;
```

□

Exercice 2 (4 points):

On souhaite modéliser le découpage en sections, sous-sections et sous-sous-sections d'un document. On utilise pour cela une structure d'arbre, où chaque noeud correspond à une section, identifiée par son titre (une chaîne), et auquel on associe la liste de ses sous-sections, elles mêmes pouvant avoir des sous-sections. Le type `section` permet de représenter ceci, où le constructeur `FinalSect` correspond à une section terminale dans sa hiérarchie. Un document est donc une liste de sections.

```
type section = FinalSect of string
          | Section of string * section list;;

type document = Document of section list;;

let d = Document [Section ("Amon-Re",
                [Section("Baa",
                     [FinalSect "Canope"; FinalSect "Delphes"]);
                 FinalSect "Euphrates"]);
              FinalSect "Fin"];;
```

1. Ecrire une fonction `afficheDoc` capable d'afficher la hiérarchie des sections et sousections d'un document avec un décalage à droite pour chaque nouvelle sousection. Ainsi, le document `d` de l'exemple, devra être affiché sous la forme :

   ```
   # afficheDoc d;;
   Amon-Re
     Baa
        Canope
        Delphes
     Euphrates
   Fin
   - : unit = ()
   ```

   Réponse :

   ```
   let afficheDocMarge (Document d) =
    let rec affiche d marge =
    match d
    with [] -> ()
     |  (Section (title,ss))::r ->
             let entete = margeˆtitle in
   ```

```
          print_string entete; print_newline();
          affiche ss ("␣␣"^marge);
          affiche r marge
   |  (FinalSect title)::r ->
       let entete = marge^title in
          print_string entete; print_newline();
          affiche r marge
  in affiche d "";;
```

2. Modifier la fonction précédente de manière à afficher le numéro de chaque section et sous-section. Ainsi, le document `d` de l'example, devra être affiché sous la forme :

```
# afficheDoc d;;
1  Amon-Re
  1.1  Baa
    1.1.1  Canope
    1.1.2  Delphes
  1.2  Euphrates
2  Fin
- : unit = ()
```

Réponse :

```
let afficheDocNums (Document d) =
  let rec affiche d numSec prefixe =
  match d
  with [] -> ()
   |  (Section (title,ss))::r ->
          let num = (string_of_int numSec) in
          let entete = prefixe^num^"␣"^title in
          print_string entete; print_newline();
          affiche ss 1 ("␣␣"^prefixe^num^".");
          affiche r (numSec+1) prefixe
   |  (FinalSect title)::r ->
          let num = (string_of_int numSec) in
          let entete = prefixe^num^"␣"^title in
          print_string entete; print_newline();
          affiche r (numSec+1) prefixe
  in affiche d 1 "";;
```

□