Département d'Informatique
LICENCE 2ème année
Jeudi 25 janvier 2007

Contrôle terminal de Programmation Fonctionnelle

Durée : 2h. Documents autorisés : Notes personnelles de cours, de TD et de TP. Utiliser le langage OCaml sans ses éléments impurs.

I Récursivité

On considère une grille de mots croisés à N lignes et M colonnes. Chaque case est repérée par ses coordonnées (i, j), où $i \in 1..N$ et $j \in 1..M$. On utilise les listes avec leur notation prédéfinie en OCaml. On suppose définies les fonctions `hd_string` et `tl_string` qui, étant donnée une chaine de caractères, retourne une chaine qui contient respectivement le premier caractère et les caractères suivants.

```
hd_string "TRUC" = "T"   tl_string "TRUC" = "RUC"   tl_string "C" = ""
```

1. Définir la fonction `spell` qui, étant donné un mot horizontal de la grille et les coordonnées de sa première lettre, retourne la liste des lettres de ce mot avec leurs coordonnées :

   ```
   spell ((5,1),"SAC") = [((5,1),"S");((5,2),"A");((5,3),"C")]
   ```

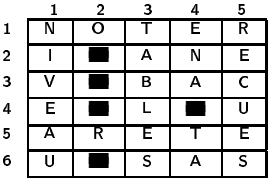
2. Définir la fonction `scan_h` qui, étant donné les dimensions N et M de la grille, retourne la liste de ses cases dans l'ordre de leur parcours. On choisit de parcourir les cases horizontalement.

   ```
   scan_h (3,2) = [(1,1);(1,2);(2,1);(2,2);(3,1);(3,2)]
   ```

II Schémas de programmes

On considère des solutions de grilles de mots-croisées. Une solution est donnée par une liste des mots horizontaux. Par exemple :

```
let soluce =
  [((1,1),"NOTER");((2,1),"I");((2,3),"ANE");
   ((3,1),"V");((3,3),"BAC");((4,1),"E");
   ((4,3),"L");((4,5),"U");((5,1),"ARETE");
   ((6,1),"U");((6,3),"SAS")];;
```



On suppose prédéfinis la concaténation de 2 listes, notée avec le symbole `@` infixé, le schéma `List.map` et le schéma `List.fold_right` vérifiant :

```
List.map f [a1; a2; ...; an] = [f a1; f a2; ...; f an]
List.fold_right f [a1; ...; an] b = f a1 (f a2 (... (f an b) ...))
```

1. Définir la fonction `scan_v` identique à la fonction `scan_h` de la question I.2 excepté que l'on choisit de parcourir les cases verticalement.

   ```
   scan_v (2,3) = [(1,1);(2,1);(1,2);(2,2);(1,3);(2,3)]
   ```

   Il est imposé de se servir de la fonction `scan_h` et du schéma `List.map` en remarquant que le résultat de `scan_v (2,3)` est identique à celui de `scan_h (3,2)` excepté que les coordonnées à l'intérieur de chaque couple sont données dans le désordre ((j,i) au lieu de (i,j)).
2. Définir la fonction `letters` qui étant donnée une solution de grille de mots-croisés retourne la liste de toutes ses lettres avec leur coordonnées.

   ```
   letters soluce =
     [((1,1),"N");((1,2),"O");((1,3),"T");((1,4),"E");((1,5),"R");
      ((2,1),"I");((2,3),"A");((2,4),"N");((2,5),"E");((3,1),"V");
      ((3,3),"B");((3,4),"A");((3,5),"C");((4,1),"E");((4,3),"L");
      ((4,5),"U");((5,1),"A");((5,2),"R");((5,3),"E");((5,4),"T");
      ((5,5),"E");((6,1),"U");((6,3),"S");((6,4),"A");((6,5),"S")]
   ```

   Vous devez utiliser la fonction `spell` et les schémas de programmes `List.map` et `List.fold_right`.
3. Définir la fonction `size` qui étant donnée une solution de grille de mots-croisés retourne les dimensions N et M de la grille.

   ```
   size soluce = (6,5)
   ```

   Vous devez utiliser la fonction `letters` et le schéma `List.fold_right`.
4. En déduire la fonction `count_holes` qui étant donnée une solution de grille de mots-croisés retourne le nombre de cases noires de la grille.

   ```
   count_holes soluce = 5
   ```

III Lambda-calcul

Réduire ces λ-expressions en utilisant l'ordre normal de réduction (NOR). Indiquer le redex avec le symbole $\curlywedge$. Préciser la règle utilisée (α- ou β-conversion) et utiliser la α-conversion uniquement lorsque cela est strictement nécessaire (en cas de problème de capture de variable).

1. $\lambda n.(n\ \lambda t.(t\ \lambda x.\lambda y.y\ a)\ \lambda x.\lambda y.y)\ \lambda f.\lambda a.a$
2. $\lambda n.(n\ \lambda t.(t\ \lambda x.\lambda y.y\ a)\ \lambda x.\lambda y.y)\ \lambda f.\lambda a.(f\ a)$

IV Compilation

Exprimer ces λ-expressions en utilisant la notation de De Bruijn :

1. $\lambda x.\lambda y.\lambda z.((z\ y)\ x)$
2. $\lambda x.\lambda y.(\lambda z.(z\ y)\ x)$