Algorithmique-Programmation : IAP1 - Rattrapage - septembre 2010

Sans documents - durée : 1h30 -

Calculette : non autorisée et sans objet - Ordinateur interdit

4 pages de sujet

Les exercices sont indépendants. Ne vous bloquez donc pas sur une question.

Pour répondre à certaines questions, il se peut que vous ayez besoin de définir des fonctions auxiliaires. Dans ce cas indiquez clairement ce que font ces fonctions auxiliaires en donnant par exemple leur interface.
Pour les fonctions explicitement demandées dans l'énoncé, n'écrivez leur interface que si celle-ci est explicitement demandée dans l'énoncé.

Exercice 1 (**Session à compléter**)

Donner les réponses de l'interprète OCaml pour chaque phrase. On suppose que les phrases sont entrées dans cet ordre. Si une phrase est mal typée, indiquez *erreur de typage* et expliquez pourquoi en quelques mots. Inutile de recopier les phrases, donnez la réponse de Ocaml (au moins type et valeur) en face du numéro de la phrase.

```
1# let ff a = if (a mod 2) = 0 then a else a+1;;

2# [ff 3 ; ff 2];;

3# let tt (f, x) = if (f x) then x else x+1;;

4# tt ((function x -> (x mod 2) = 0), 1);;

5# let uu f g x =  if (f x) then x else g x;;

6# uu (function y -> y < 0) ff 3;;

7# let mafonct = uu (function y -> y < 0) ff ;;

8# mafonct 3;;
```

Exercice 2

Compléter les interfaces des fonctions `mystere` et `mystere_aux` définies ci-dessous. Si la fonction ne lève pas d'exception, vous n'écrirez rien en face de `raises`. Si la fonction ne calcule rien, on écrira true comme post-condition. De même si la fonction n'a pas de pré-condition particulière, on écrira true.

```
exception Ex of int;;

(* interface mystere_aux
 type :
 args :
 precondition :
 postcondition :
 raises :                                  *)
let rec mystere_aux l n = match l with
 [] -> raise (Ex n)
| _::r -> mystere_aux r (n+1);;

(* interface mystere
 type :
 args :
 precondition :
 postcondition :
 raises :                                  *)
let mystere l = try mystere_aux l 0 with
                Ex p -> p;;
```

Exercice 3

On représente une personne par un couple dont le premier composant est son nom (une chaîne de caractères de type string), le second son âge (un entier de type int).

Question 1

Écrire une fonction `majeure` qui teste si une personne est majeure (i.e. dont l'âge est supérieur ou égal à 18). Elle retournera true si la personne est majeure, false sinon.

Question 2

Écrire une fonction `exist_majeur` qui prend en paramètre une liste de personnes et qui retourne `true` s'il existe au moins une personne majeure dans la liste, `false` sinon.

Exercice 4

Dans cet exercice on étudie différentes notions d'*inclusion* de deux listes.

Dans les questions suivantes, pour vérifier l'appartenance d'un élément à une liste vous pourrez utiliser sans la définir la fonction `mem` de type `'a -> 'a list -> bool`. `mem x l` vaut `true` si `x` appartient à `l`, `false` sinon.

1. On dit qu'une liste `l1` est incluse dans une liste `l2` si tous les éléments de `l1` sont des éléments de `l2`. L'ordre dans lequel les éléments apparaissent dans `l1` n'est pas nécessairement identique dans `l2`. De même pour le nombre d'occurrences.

   Par exemple la liste [2;4;1] est incluse dans la liste [1;2;3;4] et la liste [2;1;4;1] est incluse dans la liste [1;2;3;4]. Mais la liste [2;4;1] n'est pas incluse dans la liste [1;2;3].

   Question 3

   Écrire la fonction `inclus` qui prend deux listes en argument et retourne `true` si la première est incluse dans la seconde et `false` sinon.

   Vous en donnerez une version directe (récursive, utilisant le filtrage) ainsi qu'une version utilisant la fonctionnelle `forall` rappelée en fin d'énoncé.

2. On dit qu'une liste `l1` est un préfixe d'une liste `l2` si `l2` peut être définie comme la concaténation de `l1` et d'une autre liste. Ainsi tous les éléments de `l1` apparaissent en tête de `l2`, dans le même ordre et avec le même nombre d'occurrences. Une liste est préfixe d'elle-même et la liste vide est préfixe de toute autre liste.

   Par exemple la liste [1;2;4;1] est un préfixe de la liste [1;2;4;1;3;5;7]. Mais la liste [1;2;4;1] n'est pas un préfixe de la liste [1;2;4;3;5;7]. De même la liste [1;4;3] n'est pas un préfixe de la liste [1;2;4;3;5;7].

   Question 4

   Écrire la fonction `prefixe` qui prend deux listes en argument et retourne `true` si la première est un préfixe de la seconde et `false` sinon.

3. On dit qu'une liste `l1` est une sous-liste d'une liste `l2` si tous les éléments de `l1` apparaissent dans `l2` dans le même ordre mais d'autres éléments peuvent être intercalés entre ceux-ci dans `l2`. Une liste `l` est une sous-liste d'elle-même et la liste vide est une sous-liste de toute autre liste.

   Par exemple la liste [1;2;4;1] est une sous-liste de la liste [0;**1**;3;4;**2**;9;**4**;**1**;3;5;7]. Mais la liste [1;2;4] n'est pas une sous-liste de la liste [0;1;0;4;2;3;5;7] (car il manque le 4 final).

   Question 5

   Écrire la fonction `sousliste` qui prend deux listes en argument et retourne `true` si la première est une sous-liste de la seconde et `false` sinon.

   Question 6

   Quelles relations existe-t-il entre ces 3 fonctions ?

Question 7

Modifier la fonction `sousliste` de manière à ce qu'elle retourne un couple résultat de la forme `(b,l)` tel que :

- si la première liste `l1` est une sous-liste de la seconde liste `l2` alors `b` vaut `true` et `l` est la liste des positions dans `l2` des différents éléments de `l1`.
- si `l1` n'est pas une sous-liste de `l2` alors `b` vaut `false` et `l` la liste vide.

On appellera `sl_positions` cette nouvelle fonction.

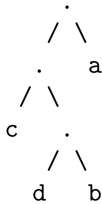
Ainsi

```
#sl_positions [1;2;4;1] [0;1;3;4;2;9;4;1;3;5;7];;
- : bool * int list = (true, [2;5;7;8])

#sl_positions [1;2;4] [0;1;0;4;2;3;5;7];
- : bool * int list = (false, [])
```

Exercice 5 (**Arbres**)

On travaille dans la suite avec des arbres binaires dont les valeurs, ici des caractères, sont rangées aux feuilles, par exemple l'arbre suivant :



On définit le type de ces arbres binaires de la façon suivante :
`type arbre = Feuille of char | Noeud of arbre * arbre`
Un arbre est soit une feuille contenant un caractère soit un arbre composé d'un sous-arbre gauche et d'un sous-arbre droit.

Question 8

Définir l'arbre exemple précédent en Ocaml. On l'appellera `ex1`.

Question 9

Écrire la fonction `feuilles` qui prend un arbre et retourne la liste des valeurs stockées aux feuilles de cet arbre. On suivra un parcours gauche droite.

Appliquée à l'arbre exemple, la fonction calcule la liste ['c';'d';'b';'a'].

Question 10

Écrire la fonction `laplusgauche` qui prend un arbre et retourne la valeur de la feuille la plus à gauche. Appliquée à l'arbre exemple, la fonction calcule 'c'.

Question 11

Écrire la fonction `laplusprofonde` qui prend un arbre et retourne la valeur de la feuille la plus profonde dans l'arbre (il y en a deux, on choisira celle de gauche par exemple). Appliquée à l'arbre exemple, la fonction calcule 'd'.

Ces arbres, complétés par une liste de déplacements (gauche, droite) peuvent servir à coder des textes (des chaînes de caractères). L'algorithme de décompression prend un arbre et une liste de déplacements (une

liste de type `dep list`, voir la définition de `dep` ci-desous) et retourne le texte correspondant, codé ici comme une liste de caractères. Il suit le fonctionnement suivant : *on part de la racine de l'arbre et on suit le chemin indiqué par les symboles D ou G (pour droite et gauche) jusqu'à arriver à une feuille, le caractère associé à cette feuille est alors ajouté au résultat et on continue de même avec le reste de la liste des déplacements en repartant de la racine de l'arbre.*
On appelle code la liste des déplacements.
Si on donne le code [D;G;D;D;D;G;G;D] avec l'arbre binaire exemple, on obtient le texte ['a';'b';'a';'c';'a'].

Question 12

```
type dep = G | D;
```

Écrire une fonction `decoder_car` qui prend en paramètre un code et un arbre et retourne le premier caractère encodé et le reste du code (sous forme d'un couple).
Par exemple avec le code [D;G;D;D;D;G;G;D] et l'arbre exemple la fonction retournera le caractère 'a' et le code [G;D;D;D;G;G;D]. Avec ce dernier code et l'arbre exemple, on obtiendra le caractère 'b' et le code [D;G;G;D]. La fonction échouera si le code est trop court pour arriver sur une feuille : par exemple avec l'arbre exemple et le code [G;D], la fonction échoue (après ces deux déplacements on n'aboutit pas à une feuille).

Question 13

En utilisant la fonction précédente, écrire la fonction `decoder` qui prend en paramètre un code et un arbre et retourne le texte encodé. La fonction pourra échouer : vous déterminerez les cas d'erreurs.

Rappel : fonctionnelles classiques sur les listes :

```
let rec map f li = match li with
  [] -> []
| x::l -> (f x)::(map f l);;

let rec filter p li = match li with
  [] -> []
| x::l -> if (p x) then x::(filter p l) else (filter p l);;

let rec fold_right f l e = match l with
  [] -> e
| a::r -> f a (fold_right f r e);;

let rec fold_left f e l = match l with
  [] -> e
| a::r -> fold_left f (f e a) r;;

let rec forall p l = match l with
   [] -> true
 | e::r -> (p e) && forall p r;;
(** forall p l vaut true si tous les éléments de l satisfont le
    prédicat p, false sinon. **)
```