Examen de programmation fonctionnelle et logique

Mardi 14 septembre 1999, 9h00-12h00

Aucun document n'est autorisé

Remarques et commentaires :

- Commencez par lire le sujet en entier, y compris les « remarques et commentaires ».
- Les exercices sont indépendants les uns des autres. Les questions au sein d'un même exercice sont elles aussi indépendantes. Vous pouvez très bien ne pas savoir répondre à une question et réussir les suivantes.
- Écrivez lisiblement et en français, car les copies seront lues (anonymat oblige) !
- Expliquez, ne serait-ce que succinctement, les codes proposés. En règle générale, facilitez leur lecture et leur compréhension.
- Les filtrages des programmes Caml doivent tous être exhaustifs.

Rappel : les itérateurs sur liste en `Ocaml`

Quatre itérateurs sur listes sont définis : `List.iter`, `List.map`, `List.fold_left` et `List.fold_right`. Les types de ces fonctionnelles et leurs comportements sont les suivants :

1. `List.iter : ('a -> 'b) -> 'a list -> unit.`
   `List.iter f [a1 ; ... ; an]` est équivalent à `begin f a1 ; ... ; f an ; () end`.
2. `List.map : ('a -> 'b) -> 'a list -> 'b list.`
   `List.map f [a1 ; ... ; an]` est équivalent à `[f a1 ; ... ; f an]`.
3. `List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a.`
   `List.fold_left f a [b1 ; ... ; bn]` est équivalent à `f ( ... (f (f a b1) b2) ... ) bn`.
4. `List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b .`
   `List.fold_right f [a1 ; ... ; an] b` est équivalent à `f a1 (f a2 (... (f an b) ...))`.

Exercice 1: Listes et sommations d'entiers (Ocaml)

Dans cet exercice, **donnez le type** de toutes les fonctions que vous écrirez !

1. Écrire une fonction qui prend en entrée deux entiers `x` et `y` et qui renvoie leur somme.
2. Écrire une fonction qui prend en entrée un entier `x` et qui renvoie une fonction qui prend en entrée un entier `y` et qui renvoie la somme de `x` et de `y`. Donner deux versions différentes de cette fonction, l'une d'entre elles réutilisant la fonction produite à la question précédente.
3. Écrire une fonction récursive qui calcule la somme des éléments d'une liste d'entiers.
4. Écrire une fonction qui calcule la somme des éléments d'une liste d'entiers **au moyen d'un itérateur sur listes**.
5. On veut décomposer une liste en deux sous-listes, de toutes les manières possibles. Écrire une fonction qui prend en entrée une liste `l` et qui retourne une liste de toutes les « paires de listes » `[a ;b]` telles que `l` soit égale à la réunion des listes `a` et `b`. Nous appelons ici « paire de listes » une liste de deux listes. Exemple :

```
#touteslespaires [1;2];;
 - : int list list list = [[[2; 1]; []]; [[1]; [2]]; [[2]; [1]]; [[]; [2
; 1]]]
```

N.B. : l'ordre des éléments dans les différentes listes est sans importance.

6. Écrire une fonction qui prend en entrée une liste contenant deux listes d'entiers, notées `x` et `y`, et qui renvoie un booléen vrai si et seulement si les listes `x` et `y` sont de même somme.
7. Écrire une fonction qui prend en entrée une liste `l` et qui renvoie un booléen, booléen qui est vrai si et seulement la liste `l` peut être décomposée en deux sous-listes de même somme. Ici on n'utilisera pas d'itérateurs, mais des fonctions récursives.
8. Même question en utilisant des itérateurs sur listes mais pas de récursivité (on peut bien sûr réutiliser des fonctions écrites pour répondre aux questions 1 à 6).

Rappel : dans cet exercice, vous devez donner le type de toutes les fonctions que vous avez écrites.

Exercice 2: Produit optimisé d'entiers (Ocaml)

Nous disposons d'une liste d'entiers dont nous voulons calculer le produit. Comme le produit d'une ensemble d'entiers est nul si l'un d'entre eux est nul, nous voulons, au moyen d'exceptions, arrêter le calcul et renvoyer la valeur du produit dès qu'un élément nul est rencontré.

1. Écrire une fonction qui calcule un tel produit optimisé en utilisant la récursivité mais pas d'itérateurs.
2. Écrire une fonction qui calcule un tel produit optimisé en n'utilisant pas la récursion.

Exercice 3: Le compte est bon (Prolog)

Le but de cet exercice est de réaliser un prédicat qui prend en premier argument une liste d'entiers, en deuxième argument un entier, appelé « cible », et en troisième argument une chaîne de caractères. Ce prédicat doit être vrai si et seulement si l'entier cible peut-être obtenu à partir de la liste d'entiers et d'opérations arithmétiques élémentaires, comme décrit par la chaîne de caractères. Chaque élément de la liste peut être utilisé au plus une fois. Exemple :

```
45 ?- valeursol([2,6,7,25,10],340,S).
S = '(((6*7)-25)*(2*10))'
Yes
```

Nous disposons dans cet exemple des nombres : 2, 6, 7, 25 et 10. Nous voulons savoir si en les combinant au moyen d'opérations arithmétiques élémentaires (addition, soustraction, multiplication et division), il est possible d'obtenir 340. La réponse est positive puisque $340 = (6 * 7 - 25) * 2 * 10$.

1. Écrire un prédicat qui prend en entrée trois listes et qui est vrai si la première liste est la réunion des éléments des deux autres listes. Exemple :

```
46 ?- subd([6,7],P,Q).
P = [6, 7]
Q = [] ;
P = [6]
Q = [7] ;
P = [7]
Q = [6] ;
P = []
Q = [6, 7] ;
No
```

2. Écrire un prédicat qui prend en entrée un unique argument et qui est vrai si et seulement si cet argument n'est pas la liste vide.
3. Écrire un prédicat qui décompose une liste en deux sous-listes non vides. Exemple :

```
46 ?- ssubd([6,7],P,Q).
P = [6]
Q = [7] ;
P = [7]
Q = [6] ;
No
```

4. Écrire un prédicat à un seul argument, et qui est vrai si et seulement si son argument n'est pas l'entier zéro.
5. Écrire un prédicat qui prend en entrée trois entiers et qui est vrai si et seulement si son troisième argument peut-être obtenu à partir des deux premiers et d'une des quatre opérations élémentaires : addition, soustraction, multiplication et division.

   Attention : la division est ici une division entre entiers et il est interdit de diviser par zéro. L'opérateur du modulo se note « `mod` » et celui de la division « `//` ». Tous les deux s'utilisent sous la forme « `A mod B` ».
6. Écrire un prédicat qui résout le problème, sans tenir compte de la chaîne de caractères décrivant la suite des opérations. Ce prédicat prend donc en entrée une liste d'entiers et un entier cible. Il est vrai si la cible peut-être obtenue à partir des éléments de la liste. Schématiquement, le prédicat est vrai si la cible est élément de la liste, ou si elle peut être obtenue à partir d'entiers qui peuvent être obtenus à partir de la liste.
7. Nous allons maintenant reprendre les prédicats précédemment élaborés et leur intégrer la construction de la chaîne de caractères décrivant le calcul.

   Écrire un prédicat qui prend en entrée quatre chaînes de caractères. Les deux premières chaînes représentent des nombres ; la troisième représente une opération élémentaire et la quatrième représente la combinaison des deux premières au moyen de la troisième. Exemple :

   ```
   47 ?- chaine('5','(3+4)','*',R).
   R = '(5*(3+4))'
   Yes
   ```

8. Réécrire le prédicat de la question 5 pour qu'il produise une chaîne de caractères codant l'opération réalisée par le prédicat. Le prédicat demandé ici prend donc en argument trois chaînes de caractères : les deux qui codent les nombres à combiner et celle qui code l'opération effectuée.
9. Écrire un prédicat qui résolve le problème du « compte est bon ».