Programmation fonctionnelle 2024-2025

Feuille 11 : Efficacité

Exercice 11.1 1. On considère la fonction suivante qui concatène deux listes :

```
let rec append l1 l2 =
  match l1 with
    [] -> l2
  | h :: t -> h :: append t l2
```

Combien d'appels récursifs l'appel `append l1 l2` provoque-t-il ? Donnez votre réponse en fonction de la longueur des listes `l1` et `l2`.

2. Combien de fois l'opérateur `::` est-il utilisé sur l'appel `append l1 l2` ?

3. On utilise la fonction `append` pour écrire la fonction `reverse` suivante, qui renverse une liste :

```
let rec reverse l =
  match l with
    [] -> []
  | h :: t -> append (reverse t) [h]
```

On appelle $c(n)$ le nombre d'utilisations de l'opérateur `::` sur l'appel `reverse l`, où n est la taille de la liste `l`. Remarquer que ce nombre dépend seulement de la longueur de la liste `l` (et pas des valeurs des éléments de la liste). En utilisant la question précédente et la définition récursive de `reverse`, écrire une récurrence satisfaite par $c(n)$.

4. Déduire de la question précédente la valeur de $c(n)$.

Exercice 11.2 Dans cet exercice, on veut écrire une fonction `reverse_efficace` telle que `reverse_efficace l` utilise n fois l'opérateur `::`, où n est la longueur de la liste `l`.

1. Que calcule la fonction suivante ? Justifiez votre réponse.

```
let rec rev_append l acc =
  match l with
    [] -> acc
  | h :: t -> rev_append t (h :: acc)
```

2. En utilisant la fonction `rev_append` de la question 1, écrivez une fonction `reverse_efficace` telle que `reverse_efficace [a1;...;an]` calcule la liste `[an;...;a1]`.

3. Montrez que `reverse_efficace l` effectue bien n utilisations de l'opérateur `::`, où n est la longueur de `l`.

Exercice 11.3 1. Définir un type `e_b_c` (pour **e** pour entier, **b** pour booléen, **c** pour chaîne) permettant de représenter soit un entier, soit un booléen, soit une chaîne de caractères.

2. Écrire une fonction `somme` qui prend en argument une liste d'éléments de type `e_b_c` et calcule la somme de tous les entiers de cette liste (les autres éléments de la liste seront ignorés).

3. Écrire une version récursive terminale de la fonction `somme` précédente.

4. Écrire une fonction `filtre_int` qui prend en argument une liste d'éléments de type `e_b_c` et calcule

la liste de tous les entiers de cette liste (les autres éléments de la liste seront ignorés).

5. Écrire une version récursive terminale de la fonction `filtre_int` précédente.
6. Écrire une fonction `concat` qui prend en argument une liste d'éléments de type `e_b_c` et calcule la concaténation de toutes les chaînes de cette liste.
7. Écrire une version récursive terminale de la fonction `concat` précédente.

Exercice 11.4 On veut écrire une version récursive terminale de la fonction `power`. On considère la fonction $g(a, x, n) = ax^n$.

1. Écrire une équation liant $g(a, x, n)$ et $g(a', x', n-1)$, pour $n > 0$ et a', x' bien choisis.
2. En déduire une fonction `power` récursive terminale.
3. Reprendre les 2 questions précédentes en
   - écrivant une récurrence liant $g(a, x, n)$ et $g(a', x', n/2)$,
   - écrivant une fonction `power'` récursive terminale et plus efficace que la fonction `power` ci-dessus.
4. Combien d'appels récursifs sont-ils effectués dans le premier cas, en fonction de l'exposant n ? Dans le second ?
5. Écrire une version générique `power_gen mult one x n` de cette fonction, de type

   `('a -> 'a -> 'a) -> 'a -> 'a -> int -> 'a`,

   où `mult` est une fonction de multiplication, et qui calcule la puissance `n`-ème de `x` (pour cette multiplication) multipliée par `one`.

Exercice 11.5 La suite de Fibonacci est définie par $F_0 = 0$, $F_1 = 1$ et $F_n = F_{n-1} + F_{n-2}$ pour $n > 1$.

1. Écrire une fonction récursive naïve `fib1: int -> int` telle que `fib1 n` calcule F_n.
2. Montrer que le nombre d'additions effectuées par `fib1 n` est $2^{\Theta(n)}$.
3. Écrire une version `fib2 n`
   - récursive terminale,
   - et effectuant seulement $\Theta(n)$ additions.

   Aide : considérer la *suite de Fibonacci généralisée* définie par $G_0 = a$, $G_1 = b$ et $G_n = G_{n-1} + G_{n-2}$ pour $n > 1$, où a et b sont deux entiers naturels quelconques.
4. Soit F la matrice suivante :
   $$F = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}.$$
   Montrer que pour tout $n > 0$, on a :
   $$F^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$
5. Utiliser la fonction `power_gen` de l'exercice précédent pour écrire une fonction `fib3` telle que `fib3 n` calcule F_n en effectuant $\Theta(\log n)$ opérations arithmétiques (addition ou multiplication d'entiers).

Exercice 11.6 1. Écrire une fonction `decomp10` qui a un entier `n` associe la liste des chiffres de sa décomposition en base 10, chiffre de poids faible en tête. Par exemple, `decomp10 239847` doit retourner `[7; 4; 8; 9; 3; 2]`.

2. Écrire la fonction réciproque, qui prend en argument une liste de chiffres de 0 à 9, et qui retourne l'entier codé par cette liste (chiffre de poids faible en tête).
3. Reprendre les questions 1 et 2 en remplaçant chiffre de poids faible par chiffre de poids fort.
4. Que faut-il changer aux questions précédentes pour traiter les mêmes questions en base 2 ?