Programmation fonctionnelle : feuille d'exercices 2

María-Virginia Aponte

15 mars 2011

Exercice 1

Récursivité
Considérez la fonction récursive suivante.

```
# let rec sommeN n =
     if n<=0  then 0 else n + sommeN(n-1);;
val sommeN : int -> int = <fun>
```

1. Testez cette fonction en Ocaml pour les appels suivants : `sommeN(0)`, `sommeN(-1)`, `sommeN(1)`, `sommeN(3)`.
2. Déroulez manuellement ces appels. *Rappel* : un déroulement récursif peut se faire manuellement en remplaçant chaque appel à la fonction par son corps où le paramètre formel est remplacé par le paramètre de l'appel, puis en réalisant les opérations correspondantes (en mimant l'exécution). Par exemple, pour la factorielle et l'appel `fact(3)` cela donne :

```
# let rec fact n =
     if n<=1 then 1  else n*fact (n-1);;

fact(3) =
   =  if 3<=1 then 1  else 3*fact (3-1)
   =  3 * fact (2)                                  (* car 3>1 *)
   =  3 * (if 2<=1 then 1  else 2*fact (2-1))
   =  3 * (2 * fact (1))                            (* car 2>1 *)
   =  3 * (2 * (if 1<=1 then 1  else 2*fact (1-1)))
   =  3 * (2 * (1))                                 (* car 1 <= 1 *)
   =  3 * 2 * 1
   =  6
```

3. Cette fonction termine-t-elle dans tous les cas ? Pouvez vous dire pourquoi ?

Exercice 2

Terminaison
Considérez les définitions de fonctions suivantes. Ces fonctions terminent-elles toujours ? Donnez un déroulement récursif manuel pour les appels donnés plus bas, puis testez ces appels dans la boucle intéractive d'Ocaml.

1. ```
   # let rec fact n =
         if n=1 then 1 else n*fact(n-1);;
   ```
   Testez cette fonction avec les appels fact 0 et fact (-1). Quel est le problème ? Donnez une correction simple.
2. ```
   # let rec fact n =
         if n<=1 then 1 else n*fact(n+1);;
   ```
   Testez cette fonction avec les appels (fact 0) et (fact 2). Quel est le problème ? Donnez une correction simple.

3. `# let rec f n = f(n-1);;`
   Testez cette fonction avec l'appel f 0. Quel est le problème ?
4. `# let rec g n =`
   `    if n<=0 then g(n-1) else g(n+1);;`
   Testez cette fonction avec les appels g 0 et g 1. Quel est le problème ?

Exercice 3

Programmation, récursivité
On souhaite écrire une fonction récursive qui calcule la puissance x^b, pour $x, b \geq 0$.

1. Donnez une définition récursive dans un style mathématique pour cette fonction. *Rappel* : voici la définition mathématique de la factorielle :

$$n! = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \\ n \times (n-1)! & \text{si } n > 0 \end{cases}$$

2. Ecrivez cette fonction en Ocaml.
3. Testez votre fonction. Termine-t-elle toujours ? Pourquoi ?

Exercice 4

Récursivité, programmation, fonctions d'ordre supérieur
Considérez les fonction suivantes :

```
let lettre c = ('a' <= c && c<= 'z') or ('A'<=c && c<= 'Z');;

let rec motI (s,i) =
   if i<=0 then true
   else (lettre (s.[i])) && motI (s,(i-1));;
val motI : string * int -> bool = <fun>

# motI ("abcd456",3);;
- : bool = true

# motI ("abcd456",5);;
- : bool = false
```

Question 1

1. Testez en Ocaml les exemples proposés plus haut. Donnez un déroulement récursif de l'appel `motI ("abcd456",3)` et de l'appel `motI ("abcd456",5)`
2. Donnez une définition mathématique de cette fonction. Que fait la fonction `motI` ? Quel est le rôle du paramètre i ?
3. On souhaite écrire une fonction qui teste si une chaîne est composée uniquement de lettres. Proposez une solution qui incorpore les fonctions `lettre` et `motI` en tant que fonctions locales. Cette dernière possède 2 arguments mais un seul change pendant les appels récursifs. Utilisez le schéma récursif 2 décrit dans les transparents du cours.
4. Que fait la fonction suivante ? Comparez la avec la fonction que vous avez écrite. Comment éffectue-t-elle le parcours récursif de la chaîne ? Comparez ce parcours avec celui effectué par `motI`.

```
let  est_mot s =
 let lettre c = ('a' <= c && c<= 'z') or ('A'<=c && c<= 'Z') in
 let rec parcours i =
   if i>= String.length s then true
   else (lettre (s.[i])) && parcours (i+1)
 in parcours 0;;
val est_mot : string -> bool = <fun>

# est_mot "abc";;
- : bool = true

# est_mot "687678";;
- : bool = false
```

5. Inspirez vous de l'écriture compacte de la fonction `palindrome` dans les transparents du cours afin de donner une version plus courte des fonctions `parcours` et `motI` en utilisant des opérateurs logiques plutôt qu'une conditionnelle.
6. La directive `trace` permet de tracer tous les appels déclenchés par une fonction récursive. Cela ne marche pas pour les fonctions récursives locales. Nous allons donc définir `parcours` globalement afin de tester plusieurs appels. Attention : il faut déclarer la fonction à tracer par `# trace nom-fonction`, où l'on écrit explicitement un caractère #

```
# let lettre c = ('a' <= c && c<= 'z') or ('A'<=c && c<= 'Z') ;;
val lettre : char -> bool = <fun>

#  let rec parcours i =
       if i>= String.length s then true
       else (lettre (s.[i])) && parcours (i+1);;
    val parcours : int -> bool = <fun>

# #trace parcours;;
parcours is now traced.

# parcours 0;;
```

Question 2

Inspirez-vous de la fonction précédente pour écrire la fonction qui transforme une chaîne de caractères composée uniquement de chiffres en un nombre entier. *Indication* : vous pourrez convertir chaque caractère vers son correspondant en chiffre entier, puis multiplier celui-ci par la puissance de 10 correspondant à son indice dans la chaîne. La somme de toutes ces opérations correspondra alos au résultat final. Exemple : le résultat à envoyer pour "283" est obetnue par $2 \times 10^2 + 8 \times 10^1 + 3 \times 10^0 = 200 + 80 + 3 = 283$.

Question 3

1. Ecrivez une fonction `testeChaine` qui prend une chaîne s et une fonction de test p et teste si tous les caractères de s satisfont le test p. Quel est le type de votre fonction ?
2. Réecrivez la fonction `est_mot` par un simple appel à la fonction `testeChaine`.
3. Ecrivez une fonction `est_nombre` qui teste si une chaîne ne contient que des nombres par un appel à `testeChaine`.

Exercice 5

Polymorphisme, typage.
Pour les fonctions suivantes, tentez de découvrir leur type, puis comparez votre réponse avec celle données par Ocaml.

```
# let f(x,y) = (y,x);;

# let f(x,y) = x^y;;

#  let f(x,y) = (x+y, x-y, x*y);;

# let f(x,y) = (x+y,  x^y);;

# let f x = fst x;;

# let f(x,y) = fst x +  fst y;;

# let f(x,y) = (x, fst y);;

# let f(x,y) = (fst x, fst y);;

# let g x y = (x,y);;

# let g x y = (x,y,x&&y);;
```

Exercice 6

Filtrage.
Lorsqu'il y a erreur, expliquez les messages affichés par Ocaml, et proposez une solution. Pour les fonctions sans erreur, testez-les sur plusieurs exemplez et expliquez ce qu'elles font.

```
# let  voyelle c =
 match c
 with 'a'| 'e' | 'i' | 'o' | 'u' -> true
   |   _ -> false;;
val voyelle : char -> bool = <fun>

# voyelle 'b';;
- : bool = false
# voyelle 'i';;
- : bool = true

#let  opArith c =
 match c
 with    _ -> false;;
  | '+'| '-' | '*' | '/'  -> true
val opArith : 'a -> bool = <fun>

# opArith 1;;
- : bool = false
# opArith '+';;
- : bool = false

#let sommeTriple a =
 match a
 with (x,y,z) -> x+y+z
  |   (0,0)   -> -1;;
This pattern matches values of type int * int
but is here used to match values of type 'a * 'b * 'c
```

```
#let sommeTripleBis a =
 match a
 with  (0 ,0,0)   -> false
  |   (x,y,z) -> x+y+z
This expression has type int but is here used with type bool


# let reponse r =
  match r
  with"oui" -> trueue
  |     "Oui" -> true
  |     "OUI" -> true;;
Warning: this pattern-matching is not exhaustive.
Here is an example of a value that is not matched:
""
val reponse : string -> bool = <fun>

val reponse : string -> bool = <fun>
# reponse "oui";;
- : bool = true
# reponse "non";;
Exception: Match_failure ("", 2, 0).
```

Exercice 7

Définir un type enregistrement `date` pour modéliser une date composée d'un numéro de jour, d'un numéro de mois et d'une année. Ecrivez ensuite :

1. Un exemple de date dans la variable `aujourd'hui`.
2. Une fonction `bissextile` qui teste si une année est bissextile. Une année est bissextile si elle est divisible par 4, sauf si elle est disible par 100. Il existe une exception à l'exception : les années divisibles par 400 sont aussi bissextiles. *Nota bene* : l'opération `a mod b` renvoie le reste de la division entière de a par b.
3. Une fonction `joursMois` qui calcule le nombre de jours dans un mois pour une année donnée.
4. En utilisant les deux fonctions précédentes, écrire une fonction `lendemain` qui étant donné une date supposée correcte calcule la date du lendemain.

Exercice 8

Définissez un type `employe` caractérisé par un nom, un salaire, et une date d'entrée dans l'entreprise. Ecrivez des fonctions pour :

1. En prenant deux employés, renvoyer celui dont le salaire est le plus élévé.
2. Calculer l'ancienneté d'un employé en mois et en années. Exemples : 2 mois ou 1 an et 2 mois.