AFP3

November 4, 2024

1 I - Fonctions récursives sur les entiers

Exemple de la fonction `factorielle`

[1]:
```
let rec fact n =
  match n with
  | 0 -> 1
  | _ -> n * (fact (n-1))

printfn "%A" (fact 5)
```

```
120
```

Remarque: les opérateurs sont aussi des fonctions !

Et `1+2` peut aussi s écrire `(+) 1 2`

Avec la fonction factorielle:

[2]:
```
let rec fact n =
  match n with
  | 0 -> 1
  | _ -> (*) n (fact (n-1))

printfn "%A" (fact 5)
```

```
120
```

On peut *généraliser* ce schéma de récursion à l aide d une valeur v et d une fonction h:

[8]:
```
let rec schema v h n =
  match n with
  | 0 -> v
  | _ -> h n (schema v h (n-1))

printfn "%A" (schema 1 (*) 5)
```

```
120
```

L intérêt est alors de pouvoir définir plus simplement les autres fonctions basées sur le même schéma !

Remarquez aussi que ce schéma est proche d une boucle `for(i=0;i<=n;i++) {...}` des langages impératifs.

[7]:
```
printfn "%A" (schema 1  (fun n r -> 2*r)  8) (** 2^8    **)
printfn "%A" (schema [] (fun n r -> n::r) 8) (** [8..1] **)
```

```
256
[8; 7; 6; 5; 4; 3; 2; 1]
```

Remarque: le langage `F#` offre de nombreux "sucres syntaxiques" et le code précédent peut se simplifier en:

[9]:
```
let rec schema v h = function
  | 0 -> v
  | n -> h n (schema v h (n-1))

printfn "%A" (schema 1 (*) 5)
```

```
120
```

Exercises

1. Comment définir la fonction x^n ?
2. Comment définir la fonction $interval(n, m) = [n, n+1, ..., m]$?
3. Proposer un autre exemple de fonction pouvant être généré à partir du schéma précédent.

2 II - Fonctions récursives sur les listes

On peut reprendre le même raisonnement que ci-dessus sur les listes.

[11]:
```
let rec iter v f = function
  | []    -> v
  | y::ys -> f y (iter v f ys)

printfn "%A" (iter []     (fun x r -> (x+2)::r) [12;8;14]) (** map (+2) **)
printfn "%A" (iter [3;4]  (fun x r->x::r)       [1;2])     (** concat **)
printfn "%A" (iter 0      (+)                   [12;8;14]) (** sum **)
```

```
[14; 10; 16]
[1; 2; 3; 4]
34
```

Exercices

1. Comment redéfinir la fonction `map(f)` ?
2. Comment redéfinir l opérateur `(@)` ?
3. Comment définir l opérateur "big-cat" qui concaténe non plus deux listes mais une liste de liste ?
4. Comment définir la fonction `filter(p)` en utilisant `map` et la fonction précédente ?

3 III - Cas des opérateurs binaires

Il est possible de faire une *double récursion* de la manière suivante:

[27]:
```
let rec add xs ys =
  match xs,ys with
  | [],[]       -> []
  | v::vs,w::ws -> ((+) v w)::(add vs ws)
  | _           -> raise (System.ArgumentException "Lists have different sizes!
 ↪")

printfn "%A" (add [1;2] [3;4]) (** [1;2]+[3;4] **)
```

```
[4; 6]
```

Remarque: la virgule `(,)` permet de construire des n-uplets (avec des paires ci dessus).

Exercise

Comment généraliser la fonction précédente pour faire la soustraction de 2 vecteurs ou le produit terme à terme ?

[33]:
```
let rec bop v0 f xs ys = (** Binary OPerator **)
  match xs,ys with
  | [],[]       -> v0
  | v::vs,w::ws -> (f v w)::(bop v0 f vs ws)
  | _           -> raise (System.ArgumentException "Lists have different sizes!
 ↪")

let (<+>) = bop [] (+)
let (<->) = bop [] (-)
let (<*>) = bop [] (*)

printfn "%A" ([1;2] <+> [3;4])
printfn "%A" ([1;2] <-> [3;4])
printfn "%A" ([1;2] <*> [3;4])
```

```
[4; 6]
[-2; -2]
[3; 8]
```

4 IV - Modèle Dataflow (“flot de données/traitements”)

Comment utiliser l opérateur suivant dans l expression `g (f x)` ?

[37]:
```
let (|>) x f = f x
```

La programmation fonctionnelle permet d exprimer des séquences de traitement comme l illustre le code suivant.

[70]:
```
let map f = iter [] (fun x r -> (f x)::r)  (** [f x1;f x2;...; f xn] **)

let (<..>) n m =                           (** [n;n+1;...;m]          **)
  let range = schema [] (fun n r -> n::r)
  (range (m-n+1)) |> (map (fun x->m+1-x))

let sum = iter 0.0 (+)

let dataflow =
  (0 <..> 10)
  |> map float
  |> map ((*) 0.31415)
  |> map Math.Sin
  |> sum
  |> (*) 0.31415
  |> printfn "%A"
```

```
1.98353906
```

Exercises

1. Représenter graphiquement le modèle flot de données précédent et dire quels sont les traitements réalisés par les différents blocs.
2. Comment généraliser le code défini pour calculer $A = \int_a^b f(x).dx$?

5 V - Application

Les éléments qui précèdent permettent en oeuvre des modèles mathématiques “complexes” comme les *réseaux de neurones*, le *calcul matriciel*, les *régressions (linéaires)*, etc.

Le code suivant montre ainsi l utilisation d un algorithme d apprentissage pour trouver l equation de la droite $y = a.x + b$ passant par un ensemble de points (x_i, y_i).

[86]:
```
let a,b = 1.0, 2.0   (** solution to find **)
let y x = a*x+b
let dataset = [0.0..0.1..1.0] |> List.map y

let af,bf = 0.0, 1.0 (** random initialization/weights **)
let neuron a b x = a*x+b

let rec learn af bf n =
  match n with
  | 0 -> af,bf
  | _ -> let x       = dataset[n%10]
         let error   = (y x) - (neuron af bf x)
         let rate    = 0.1
         let af1,bf1 = af+rate*error,bf+rate*error
         learn af1 bf1 (n-1)
```

```
learn af bf 10 |> fun (a,b) -> printfn "Found y = %.3fx + %.3f" a b
```

```
Found y = 0.986x + 1.986
```

6 VI - Pour aller plus loin...

Comment généraliser ce qui précède pour faire une *interpolation polynomiale* ?