FP2

November 27, 2024

1 Définition de types

Il est possible de créer des types spécifiques en utilisant l'instruction `type_of` en précisant les différents "constructeurs" pour définir toutes les valeurs d'un type ; les constructeurs (constantes ou fonctions) sont séparés alors par le symbol (`|`).

1. Exemple des `Booléens`

[4]:
```
type Bool = True | False

let neg = function
  | True -> False
  | _    -> True

let (<&>) b1 b2 =
  match b1 with
  | True -> b2
  | _    -> False

neg (True <&> False)
```

```
True
```

2. Un type peut être `récursif` comme l'illustre l'exemple des entiers naturels ci-dessous:

[6]:
```
type Nat = Zero | Succ of Nat

let v0 = Zero
let v1 = Succ Zero

let rec (<+>) n1 n2 =
  match n1 with
  | Zero   -> n2
  | Succ n -> Succ (n <+> n2)

v0 <+> v1 = v1
```

3. Enfin, un type peut être "générique" en ajoutant une "variable de type". Il est ainsi possible de considérer des listes d'entiers, des listes de caractères (string), des listes de listes, etc.

[11]:
```
type 't List = Nil | Cons of 't * ('t List)

let l0 = Nil                    (** []    **)
let l1 = Cons (1,Nil)           (** [1]   **)
let l2 = Cons (2,Cons(3,Nil))   (** [2;3] **)

let rec (@) l1 l2 =
  match l1 with
  | Nil        -> l2
  | Cons (v,l) -> Cons (v,l @ l2)

printfn "%A" (l1 @ l2)
```

```
Cons (1, Cons (2, Cons (3, Nil)))
```

Exercises

1. Comment définir la fonction `map f` sur cette définition des listes ?
2. Quelle est la fonction "plus générale" sur les listes et comment l'utiliser pour définir la fonction `map` ?
3. Proposer une transformation inversible permettant de passer aux listes standard de `F#` ?

2 Les Arbres

Les Arbres Binaires de Recherche (ABR) sont composées d'une valeur `v` et de 2 sous-arbres gauche/droite. Les valeurs contenues dans le sous-arbre gauche (resp. droit) ont des valeurs inférieures ou égales (resp. supérieure) à `v`.

Les ABR ont la particularité de fournir des algorithmes plus rapide que ceux sur les listes pour notamment rechercher une valeur.

[23]:
```
type 't ABR = Node of 't * 't ABR * 't ABR | Leaf

let rec insert v = function
  | Leaf           -> Node(v,Leaf,Leaf)
  | Node (v',g,d) -> if (v<=v') then Node(v',insert v g,d) else␣
 ↪Node(v',g,insert v d)

printfn "%A" (insert 2 (insert 1 (insert 3 Leaf)))
```

```
Node (3, Node (1, Leaf, Node (2, Leaf, Leaf)), Leaf)
```

Exercises

1. Comment adapter la fonction `insert` pour pouvoir changer la relation d'ordre ?
2. Proposer une fonction inversible permettant de transformer une liste en arbre. Comment utiliser cette fonction pour trier une liste ?

3. Comment adapter la structure d'arbre pour pouvoir modéliser/coder une arborescence de fichiers ? Une page HTML ?

3 Les Dictionnaires (Map)

Un dictionnaire peut être vu simplement comme une liste "d'associations" (ou paires `(clef,valeur)`), et donc aussi comme un arbre d'associations.

Les principales opérations sur un dictionnaire sont: l'ajout/insertion d'un nouvel élément, le test pour savoir si une clef existe, et la recherche de la valeur associée à une clef.

Exercise

1. Sachant cela, comment réaliser ces 3 opérations en prenant la base de données suivantes:

[25]:
```
let db = [("Bob",12);("Kate",18);("Bill",16)]
```

2. Même question mais en remplaçant la liste par un arbre. Discuter des performances.
3. Comment utiliser ce qui précède pour réaliser un traducteur français-anglais "efficace" ?