Cours de programmation fonctionnelle et logique
Cours 7 : types somme et types produit

Table des matières

1 Type énuméré

```
#type couleur = Trèfle | Carreau | Coeur | Pique;;
 type couleur = | Trèfle | Carreau | Coeur | Pique
```

Le type `couleur` est une énumération de quatre valeurs, chacune étant définie par un constructeur de type constant.

```
#let atout = Trèfle;;
 val atout : couleur = Trèfle
```

2 Type somme

```
#type carte =   As of couleur
               | Roi of couleur
               | Dame of couleur
               | Valet of couleur
               | Petite_carte of int * couleur;;
         type carte =
   | As of couleur
   | Roi of couleur
   | Dame of couleur
   | Valet of couleur
   | Petite_carte of int * couleur
```

```
#let carte1 = As atout;;
 val carte1 : carte = As Trèfle
```

```
#let carte2 = Petite_carte(7,Pique);;
 val carte2 : carte = Petite_carte (7, Pique)
```

2.1 Filtrage

Comme pour les listes, le filtrage sur un type somme se fait par décomposition de l'objet suivant les constructeurs possibles.

```
#let valeur_d_une_carte latout = function
     As _               -> 11
   | Roi _              -> 4
   | Dame _             -> 3
   | Valet c            -> if c = latout
                           then 20
                           else 2
   | Petite_carte (10,_) -> 10
   | Petite_carte (9,c)  -> if c = latout
                           then 14
                           else 0
   | Petite_carte _ -> 0;;
                       val valeur_d_une_carte : couleur -> carte -> int
= <fun>
```

3 Type produit

```
#type monôme = {coefficient : int; degré : int};;
 type monôme = { coefficient: int; degré: int }

#let m1 = {coefficient = 1; degré = 0};;
 val m1 : monôme = {coefficient=1; degré=0}

#m1.coefficient;;
 - : int = 1

#m1.degré;;
 - : int = 0

#let de_degré_zéro = function
     {degré = 0; coefficient = _ } -> true
   |           _                   -> false;;
     val de_degré_zéro : monôme -> bool = <fun>

#let de_degré_zéro = function
     {degré = 0} -> true
   |        _         -> false;;
     val de_degré_zéro : monôme -> bool = <fun>

#let abs_monôme = function
    {coefficient = c; degré = d} as m ->
      if c < 0 then {coefficient = -c; degré = d} else m;;
      val abs_monôme : monôme -> monôme = <fun>
```