Type structuré en OCaml

Les types structurés sont équivalents aux types produits (les n-uplets) au sens où ils permettent d'agréger des informations de type différents, mais ils ont l'avantage de donner un nom à chacune des composantes, ce qui permet de s'abstraire de l'ordre des éléments de chaque type dans le n-uplet.

En OCaml on définit un type structuré en donnant son nom, puis le nom et le type de chacun de ses champs suivant la syntaxe ci-contre. Un objet de type `type_structure` est alors défini en donnant, pour chaque champ `champ_i`, une expression `e_i` de type `type_i`) qui décrit sa valeur. La syntaxe est la suivante ♣.

```
type type_structure = {
  champ_1 : type_1 ;
  champ_2 : type_2 ;
  ...
  champ_n : type_n ;
}
```

```
{champ_1 = e_1 ; champ_2 = e_2 ... ; champ_n = e_n}
```

Si s est un objet de type `type_structure`, la valeur de son `champ_i` est donnée par `s.champ_i`.

Par défaut les objets de type `type_structure` sont fixés à la création, et leur valeur n'est plus modifiable. Cependant grâce au mot clé **`mutable`** il est possible de préciser lors de la définition du type que certains champs sont modifiables (comme si c'étaient des références). Si le `champ_k` a été déclaré **`mutable`**, on accède en lecture à cette valeur par `s.champ_k` (comme dans le cas non mutable), et on peut la modifier par `s.champ_k <- new_value`, où `new_value` est une expression de type `type_k`.

Exemple :

```
1  type point_colore = {
2    x : int ;
3    y : int ;
4    mutable color : string ;
5  }
6
7  let cree_point_bleu (a:int)(b:int) : point_colore =
8    {color = "bleu"; x = a; y = b}
9
10 let est_bleu (p:point_colore) : bool =
11   p.color = "bleu"
12
13 let colorie_en_bleu (p:point_colore) : unit =
14   p.color <- "bleu"
15
16 let test_bleus : unit =
17   assert(est_bleu (cree_point_bleu 0 0));
18   assert(not (est_bleu {y=0; x=0; color = "rouge"} ));
19   let p1 = {y=0; x=0; color = "orange"} in
20   assert (not (est_bleu p1));
21   colorie_en_bleu p1;
22   assert( est_bleu p1)
```

♣ Les champs peuvent être décrits dans n'importe quel ordre, *Cf.* exemple l.8 et l.18.