Cours types récursifs et arbres

Contents

N.1) Définition inductive d'un ensemble

Idée : définir un ensemble d'objets "par récurrence".

Définition 1 [Ensemble inductif]

> Un ensemble inductif E est défini par :
> - Un ensemble K de *cas de bases.*
> - Un ensemble C de constructeurs, qui sont des fonctions $c : E^{k_c} \to E$ où $k_c \in \mathbb{N}$.
>
> On pose
> - $E_0 = K$
> - $\forall n \in \mathbb{N}^*, E_n = E_{n-1} \cup \bigcup_{c \in C} c\left(E_{n-1}^{k_c}\right)$
>
> Alors, on a $E = \bigcup_{n \in \mathbb{N}} E_n$.

En français : on peut appliquer les constructeurs autant de fois que nécessaires (en se limitant à un nombre fini) à partir des cas de base, pour obtenir tous les éléments de E.

Exemple 2

On peut définir les listes d'entiers inductivement comme :
- soit la liste vide `[]`
- soit `x::q`, où `x` est un entier et `q` une liste.

Exemple 3

On peut définir les arbres comme :
- soit une feuille `F`
- soit un noeud, composé de deux ou plus arbres !

N.2) L'équivalent OCaml : Les types récursifs

N.2.1) Types énumération

On peut écrire des types énumération en OCaml :

```
type classe =
    | MPSI_1
    | MPSI_2
    | MP_1
    | MP_2
    | MP_ET;;
```

Une valeur de type "classe" peut alors prendre une des 5 valeurs définies (et rien d'autre !).

Exemple 4

```
let ma_classe: classe = MPSI;;
```

On peut également stocker des données arbitraires dans une des options :

```
type classe =
    | MPSI_1
    | MPSI_2
    | MP_1
    | MP_2
    | MP_ET
    | Autre of str
```

Exemple 5

```
let ma_classe: classe = Autre "BCPST";;
```

La construction `match` est alors très utile, car elle permet de traiter séparément chaque cas du type énumération.

Exemple 6

```
let nb_eleves (c: classe): int = match c with
    | MPSI_1 -> 49
    | MPSI_2 -> 43
    | MP_1 -> 38
    | MP_2 -> 36
    | MP_ET -> 40
    | Autre nom -> (failwith "classe inconnue !" ^ nom)
```

N.2.2) Types récursifs

On peut également faire des types récursifs, c'est à dire, utiliser le type "classe" dans sa propre définition !

Bien sûr, il faut que certains variants n'utilisent pas de type récursif, pour qu'on puisse "s'arrêter".

```
type classe = MP | MPSI | PCSI | PC | Autre of string | DoubleNiveau of classe *
classe;;


let classe4 = DoubleNiveau (MP, PC);;

let classe5 = DoubleNiveau (DoubleNiveau (MP, PC), Autre "MPI");;
```

N.3) Petit point sur les listes

En OCaml, on pourrait définir nous même un type de liste :

```
type liste = Vide | El of int * liste;;
```