FL: Construction de types

IUT de Lens

DUT S4

Remarques sur l'égalité

- Le symbole = est le symbole d'égalité structurelle. Elle teste l'égalité de deux valeurs en explorant leur structure.
- L'égalité physique, représentée par le symbole ==, teste si les deux valeurs occupent la même zone mémoire.
- Les deux tests retournent les mêmes valeurs pour les types primitifs (entiers, caractères, booléens, etc).

Définitions de type

- La définition d'un type se fait à l'aide du mot clé-type.
- Les noms des constructeurs commencent par une majuscule.
- Un constructeur permet de construire les valeurs d'un type et d'accéder aux composantes de ces valeurs grâce au mécanisme de filtrage.

```
# type couleur = Bleu | Rouge | Vert;;
# Vert;;
- : couleur = Vert
# let valeur = function
      Bleu  -> 1
     |Vert ->2
     | Rouge -> 3;;
val valeur : couleur -> int = <fun>
```

Somme de types

- La somme de deux types est l'union disjointe.

```
# type int_ou_bool = I of int | B of bool;;
type int_ou_bool = I of int | B of bool
# I 4;;
- : int_ou_bool = I 4
# B true;;
- : int_ou_bool = B true
# B 5;;
This expression has type int
but is here used with type bool
```

Somme de types

- Une somme de types est utilisée par analyse de motifs.

```
# let f = function
      B x -> if x then 1 else 2
    | I x->x+4;;
val f : int_ou_bool -> int = <fun>
# let f' = function
     B true -> 1
    |B false->2
    |I x->x+4;;
val f' : int_ou_bool -> int = <fun>
```

Types récursifs

- On utilise les constructeurs pour définir des types récursifs.

```
# type entier = Zero | Succ of entier;;
type entier = Zero | Succ of entier
# let succ x = Succ x;;
val succ : entier -> entier = <fun>
```

Exceptions

```
# let tete = function
          [] -> failwith "tete"
        | x::_ -> x;;
val tete : 'a list -> 'a = <fun>
# failwith;;
- : string -> 'a = <fun>
# tete [];;
Uncaught exception: Failure "tete".
```

Exceptions

- En Caml, les exceptions appartiennent à un type prédéfini **exn**.
- C'est un type somme qui est extensible en ce sens que l'on peut étendre l'ensemble de ses valeurs en déclarant de nouveaux constructeurs.
- La syntaxe pour définir de nouvelles exceptions est :

```
# exception Erreur;;
exception Erreur
# Erreur;;
- : exn = Erreur
```

Exceptions

- Le constructeur **Failure** est un constructeur d'exception.

```
# Failure "Erreur !";;
- : exn = Failure "Erreur !"
```

- La levée d'une exception se fait à l'aide de la fonction **raise** :

```
#exception ListeVide;;
exception ListeVide
# let tete = function
      [] -> raise ListeVide
    | x::_ -> x;;
val tete : 'a list -> 'a = <fun>
```

- La construction **try ... with** : **with** permet de réaliser les captures.

Exceptions

```
#exception Negatif of int;;
exception Negatif of int
#let rec fact n =
 if n <0 then raise (Negatif n) else
if n=0 then 1 else n*fact(n-1);;
 val fact : int -> int = <fun>
#let afficheFactPremier l =
     try
        print_int (fact (tete l))
     with
       ListeVide -> print_string "la liste est vide"
        |Negatif x  -> print_int x;
         print_string " est negatif"
        |_          -> print_string " Autre exception";
val affichePremier : int list -> unit = <fun>
```