

## Cours 2 : Références, fermetures et polymorphisme

### 1 Références et fonctions

Il est important de bien distinguer les deux fonctions suivantes :

```
# let f = let x = ref 3 in fun () -> x ;;  
val f : unit -> int ref = <fun>  
# let g = fun () -> let x = ref 3 in x ;;  
val g : unit -> int ref = <fun>
```

La fonction (`fun () -> x`) est définie dans un environnement local où la variable `x` est liée à une certaine valeur (en l'occurrence, la valeur retournée par `ref 3`). Lorsque cette fonction est appelée, elle a besoin d'accéder à la valeur de cette variable et donc à cet environnement. La réunion d'une fonction et de l'environnement local dans lequel elle a été définie est appelée sa **fermeture**.



Les valeurs présentes dans la fermeture d'une fonction sont partagées entre les différents appels à cette fonction. Ainsi, avec les définitions ci-dessus :

```
# let a, b = f (), f () ;;  
val a : int ref = {contents = 3}  
val b : int ref = {contents = 3}  
# incr a ; !b ;;  
- : int = 4  
# let c, d = g (), g () ;;  
val c : int ref = {contents = 3}  
val d : int ref = {contents = 3}  
# incr c ; !d ;;  
- : int = 3
```

### 2 Typage des références

Le typage des références en présence de polymorphisme est délicat. Par exemple, si les trois instructions suivantes étaient autorisées par le système de types, la troisième provoquerait une erreur de segmentation.

```
# let l = ref [];;
# l := 4::!1;;
# (function [] -> 0 | x::_ -> x +. 0.5) !1;;
```

Pour remédier à ce problème, on introduit la notion de variable de type **monomorphe**. Ces variables s'écrivent à la manière des variables polymorphes, mais se distinguent par le caractère `_` avant le nom de la variable : `'_a`, `'_b`, `'_c` etc.

```
# let l = ref []
val l : '_a list ref = {contents = []}
# l := 4::!1;;
- : unit = ()
# (function [] -> 0 | x::_ -> x +. 0.5) !1;;
This expression has type float but is here used with type int
```

Après la première instruction, le type des éléments de `l` est inconnu, mais monomorphe, ce qui est représenté par la variable `'_a`. L'ajout de `4` à cette liste `l` contraint `'_a` à être le type `int`. La référence `l` est alors de type `int list ref` ce qui explique pourquoi la dernière expression est rejetée.



Les variables de type du résultat d'une application de fonction sont toujours monomorphes. Par exemple,

```
# let identite x = x;;
val identite : 'a -> 'a = <fun>
# let id = identite identite;;
val id : '_a -> '_a = <fun>
```



Même dans le cas d'une application partielle, les variables de type du résultat sont monomorphes. Par exemple,

```
# let g () x = x;;
val g : unit -> 'a -> 'a = <fun>
# let f = g();;
val f : '_a -> '_a = <fun>
```