Algorithmique fonctionnelle - Polymorphisme et ordre supérieur

18 février 2013

Inférence de type

Définition

OCaml n'est pas explicitement typé. Le langage devine le type des variables selon l'opération appliquée.

```
# let f = fun x -> x**7. +. 4.;;
```

Question : Si l'opération ne permet pas de deviner le type de la variable que se passe-t-il ?

```
let f x y = x > y;;
```

Polymorphisme paramétrique

Définition

C'est la propriété du langage à posséder des types de données génériques.
Les données sont typées lors de l'exécution selon les données passées en paramètre.
L'expression a donc plusieurs formes selon le type de données manipulées.
Ce type généralisé est noté `'a`.

Polymorphisme

Exemples

```
# let l = [ ] ; ;
val l : 'a list = [ ]

# let maxi x y =
if x > y then x else y ; ;
val maxi : 'a -> 'a -> 'a = <fun>

let rec trouve f l =
match l with
[ ] -> failwith "Aucun element"
| t : : r -> if (f t) then t else trouve f r ; ;
val trouve : ('a -> bool) -> 'a list -> 'a = <fun>
```

Polymorphisme paramétrique

It's not a bug, it's a feature

Cela permet :

- d'augmenter le niveau d'abstraction
- de créer des algorithmes génériques

Exemple

```
maxi -7 3 ; ;
maxi "janvier" "février" ; ;
```

Polymorphisme paramétrique

Attention !

Cela ne permet pas tout :
('a -> 'b) -> 'a $\neq$
('b -> 'a) -> 'a

Mais...

('a -> 'b) -> 'a $\Leftrightarrow$
('b -> 'a) -> 'b

Ordre supérieur

Rappels

- Une fonction est considérée comme une valeur.
- Toute valeur (même fonctionnelle) peut être paramètre ou résultat de fonction

Ordre supérieur

Application partielle

C'est le fait **d'appliquer** une fonction sur un nombre de paramètres **inférieur** à celui défini pour la fonction.
Le résultat de cette application est une fonction.

Ordre supérieur

Application partielle

C'est le fait **d'appliquer** une fonction sur un nombre de paramètres **inférieur** à celui défini pour la fonction.
Le résultat de cette application est une fonction.

Exemple

```
# let fois = function x -> function y -> x * y ;;
val fois : int -> int -> int = <fun>
int -> (int -> int)
# let fois7 = fois 7 ;;
fois7 3 ;;
```

Ordre supérieur

Fonctions d'ordre supérieur

Une fonction qui a pour paramètre ou qui retourne une fonction.

Exemple

```
# let rond = function f ->
        function g -> function x -> f (g x) ; ;
val rond : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
# let rond f g x = f (g x) ; ;
```

d'autres List.map, List.fold_left ....

Curryfication

Principe

Par l'application partielle, c'est le fait de transformer une fonction à n paramètres en une fonction à un paramètre retournant une fonction à $n-1$ paramètres.

Exemple

```
# let rec puiss (x,n) =
   if n = 0 then 1
   else x * (puiss (x,n-1)) ; ;
val puiss : int * int -> int = <fun>

# let rec puiss x n =
   if n = 0 then 1
   else x * (puiss x (n-1)) ; ;
val puiss : int -> int -> int = <fun>
```

Curryfication

Principe

Par l'application partielle, c'est le fait de transformer une fonction à n paramètres en une fonction à un paramètre retournant une fonction à $n-1$ paramètres.

Exemple

```
# let rec puiss (x,n) =
   if n = 0 then 1
   else x * (puiss (x,n-1)) ; ;
val puiss : int * int -> int = <fun>

# let rec puiss x n =
   if n = 0 then 1
   else x * (puiss x (n-1)) ; ;
val puiss : int -> (int -> int)= <fun>
```

Curryfication

Principe

Par l'application partielle, c'est le fait de transformer une fonction à n paramètres en une fonction à un paramètre retournant une fonction à $n-1$ paramètres.

Exemple

```
# let rec puiss (x,n) =
   if n = 0 then 1
   else x * (puiss (x,n-1)) ; ;
val puiss : int * int -> int = <fun>

# let rec puiss x n =
   if n = 0 then 1
   else x * (puiss x (n-1)) ; ;
val puiss : int -> (int -> int)= <fun>

let cube x = puiss x 3 ; ;
```

Curryfication

Principe

L'élégance et la sécurité voudraient s'abstraire du paramètre x.
Repenser alors l'écriture de la fonction

Exemple

```
# let rec puiss n x =
    if n = 0 then 1
    else x * (puiss (n-1) x) ; ;
val puiss : int -> int -> int = <fun>

# let cube = puiss 3 ; ;
val cube : int -> int = <fun>
```

Curryfication

Exercice : Décoder les signatures suivantes

- maya -> maya -> maya = <fun>
- 'a -> 'a list -> 'a list = <fun>
- 'a -> int -> 'a list -> 'a list = <fun>
- 'a list -> ('a -> 'a -> bool) -> 'a * 'a list = <fun>
- ('a -> 'b -> 'b) -> 'b -> 'a list -> 'b = <fun>