Typage et Polymorphisme

Cours 6

Emmanuel Chailloux

Spécialité Science et Technologie du Logiciel
Master mention Informatique
Université Pierre et Marie Curie

année 2013-2014

Plan du cours

- GADT en OCaml
- présentation (rapide) de Scala
  - objets
  - filtrage par motifs
  - fonctions
  - mixin (abstraction et composition)
  - système de types

Types algébriques généralisés : GADT

But: préciser le typage sur les paramètres de types

- les contraintes sur les paramètres de type peuvent changer en fonction des constructeurs
- les variables de types sont :
  - existentielles quand elles sont en position argument d'un constructeur

Syntaxe :

```
constr-decl ::= ...
  | constr-name :  typexpr  { * typexpr } ->  typexpr

type-param ::= ...
  | [variance] _
```

Exemple sur les listes

nouvelles listes : 1 paramètre de type pour distinguer liste vide et liste nonvide, l'autre paramètre de type pour les éléments de la liste

```
1  type vide ;;
2  type pasvide ;;
3  type ('a, 'b) liste =
4    Nil : (vide,'b) liste
5  | Cons : 'b * ('a, 'b) liste -> (pasvide, 'b) liste ;;
6
7  # let l1 = Nil;;
8  val l1 : (vide, 'a) liste = Nil
9  # let l2 = Cons(3,Nil);;
10 val l2 : (pasvide, int) liste = Cons (3, Nil)
11 # let tete l = match l with Cons(x,_) -> x;;
12 val tete : (pasvide, 'a) liste -> 'a = <fun>
```

pas de *warning* sur des cas de filtrage non exhaustifs non pertinents.

Exemple : un évaluteur sans GADT

```
type expr = I of int | B of bool | Add of expr * expr
| If of expr * expr * expr;;

# let add e1 e2 =  match (e1,e2) with
  I i1, I i2 -> i1+i2
| _ -> failwith "add";;
val add : expr -> expr -> int = <fun>
# let rec eval e = match e with
    I i -> I i
  | B b -> B b
  | Add (e1,e2) -> I (add  (eval e1) (eval e2))
  | If (e1, e2, e3) -> (match (eval e1) with B b -> if b then eval e2 else ↩
       eval e3 | _ -> failwith "If") ;;
val eval : expr -> expr = <fun>

# let e1 = Add (I 3, If (B true, I 10, I 20));;
val e1 : expr = Add (I 3, If (B true, I 10, I 20))
# eval e1;;
- : expr = I 13
# let e2 = Add (I 3, B true);;
val e2 : expr = Add (I 3, B true)
# eval e2;;
Exception: Failure "add".
```

Exemple (avec GADT)

```
type 'a expr =
  I : int -> int expr
| B : bool -> bool expr
| Add : int expr * int expr -> int expr
| If : bool expr * 'a expr * 'a expr -> 'a expr ;;

# let rec eval :  type a. a expr -> a  = function e -> match e with
    I i -> i
  | B b -> b
  | Add (e1,e2) -> (eval e1) + (eval e2)
  | If (e1,e2,e3) -> if (eval e1) then (eval e2) else (eval e3);;
val eval : 'a expr -> 'a = <fun>

# let e1 = Add (I 3, If (B true, I 10, I 20));;
val e1 : int expr = Add (I 3, If (B true, I 10, I 20))
# eval e1;;
- : int = 13

# let e2 = Add (I 3, B true);;
Error: This expression has type bool expr
       but an expression was expected of type int expr
```