

Mise à Niveau et Ouverture (STL)



Emmanuel Chailloux

# Valeurs : représentation uniforme

Nécessité de parcourir les valeurs :

- ▶ Fonctions primitives génériques : égalité, sérialisation, etc.
- ▶ Gestion mémoire (cf. suite du cours)
- ▶ Introspection, affichage générique, etc.

Solution logique : **uniformiser la structure des valeurs**

Question centrale : distinction entre

- ▶ Valeurs immédiates (entiers, caractères, etc.)
- ▶ Valeurs allouées (tableaux, structures, etc.)
- ▶ Différentes sortes de valeurs allouées.

En machine : **un pointeur = un entier = un mot machine**

## Valeurs : solution plus avancées

Bit(s) discriminant(s) :

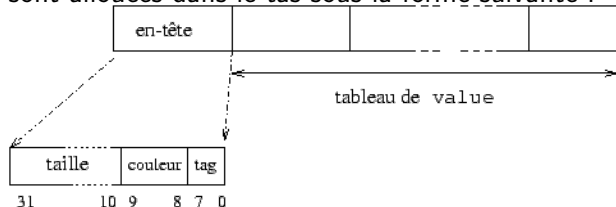
- ▶ On mange un bit sur le mot machine pour discriminer entre entier et pointeur
- ▶ Éventuellement plus de bits pour plusieurs types d'immédiats
- ▶ On utilise un système de tags comme précédemment pour les valeurs allouées
- ▶ On limite l'étendue des immédiats

En OCaml tout ce qui peut être représenté par un entier : entier, caractère, booléen, constructeur constant, est codé dans un entier et tout ce qui est plus gros qu'un entier est alloué dans le tas .

Le pointeur de la zone allouée occupe un mot mémoire, comme les entiers.

## Représentation des valeurs allouées en OCaml (1)

Les constructeurs paramétrés, les enregistrements et les fermetures sont alloués dans le tas sous la forme suivante :



le tableau de valeur correspond aux champs pour un enregistrement ou un n-uplet, ou l'environnement et le code d'une fermeture.

L'entête contient la taille du tableau de valeur, 2 bits pour le gestionnaire mémoire, et un tag correspondant au constructeur qui a alloué cette valeur.

## Représentation des valeurs allouées en OCaml (2)

On reprend la fonction `calcule_taille` pour explorer les valeurs :

```
1 # let calcule_taille o = Obj.reachable_words (Obj.repr o) ;;
2 val calcule_taille : 'a -> int = <fun>
```

On commence par des valeurs allouées plates que l'on définit spécialement :

```
1 # type ty2 = Ctor2 of int * ←
      int;;
2 type ty2 = Ctor2 of int * int
3 # let v2 = Ctor2 (3,4);;
4 val v2 : ty2 = Ctor2 (3, 4)
5 # calcule_taille v2;;
6 - : int = 3

1 # type ty5 = Ctor5 of int * int * int * ←
      int * int ;;
2 type ty5 = Ctor5 of int * int * int * int←
      * int
3 # let v5 = Ctor5 (3,6,9,12,15);;
4 val v5 : ty5 = Ctor5 (3, 6, 9, 12, 15)
5 # calcule_taille v5;;
6 - : int = 6
```

- ▶ `v2` : a 2 valeurs : 3 et 4 dans le tableau de valeur, son entête contient cette longueur de 2 est dans l'entête ainsi que le numéro de constructeur `Ctor2` dans l'octet de tag , au total : 2 valeur + 1 entête = 3 mots
- ▶ `v5` : c'est pareil mais avec un tableau de 5 valeurs = 6 au total

## Représentation des valeurs allouées en OCaml (3)

### Calcul sur une structure dynamique:

- ▶ Le constructeur `::` des listes peut être considéré comme le `Ctor2` de l'exemple précédent.
- ▶ Le constructeur constant `[]`, comme tous les constructeurs constants, est codé par un entier.

```
1 # let c = 4::[];;
2 val c : int list = [4] (* taille 3 *)
3 # let d = 22::c;;
4 val d : int list = [22; 4] (* taille 6 *)
5 # let e = 77::d;;
6 val e : int list = [77; 22; 4] (* taille 9 *)
7 # List.map calculer_taille [c;d;e];;
8 - : int list = [3; 6; 9]
```

- ▶ `c` a 2 valeurs 4 et `[]` + l'entête
- ▶ `d` prend 3 mots + la taille de `c`
- ▶ `e` prend 3 Mots plus la taille de `d`

## Pile ou tas

- ▶ size1 : utilise la pile des appels de fonction
- ▶ size2 : récursive terminale mais utilise une pile à la main

```
1 # let rec size1 (bt:'a btree) : int =
2   match bt with
3   | Empty -> 0
4   | Node(_, bt1, bt2) -> 1 + (size1 bt1) + (size1 bt2) ;;
5 val size1 : 'a btree -> int = <fun>
6
7 # let rec size_aux (bt:'a btree) (bts:( 'a btree) list) (r:int) =
8   match bt with
9   | Node (_,bt1,bt2) -> size_aux bt1 (bt2::bts) (r+1)
10  | Empty -> ( match bts with
11                | [] -> r
12                | bt::bts -> size_aux bt bts r
13                ) ;;
14 val size_aux : 'a btree -> 'a btree list -> int -> int = <fun>
15 # let size2 (bt:'a btree) : int = size_aux bt [] 0 ;;
16 val size2 : 'a btree -> int = <fun>
```

Quid pour height ?  $\Rightarrow$  lire

“Mesurer la hauteur d’un arbre” - JC Filliâtre - JFLA 2020

<https://hal.inria.fr/hal-02427360/document>