ManO (MU4IN511) : COURS 2b

Mise à Niveau et Ouverture (STL)

Emmanuel Chailloux

Représentation des données en OCaml

Polymorphisme paramétrique: représentation uniforme

- 1 mot mémoire
  - valeurs immédiates (*int*, *char*, constructeurs constants)
  - autres valeurs : pointeur vers le tas (zone d'allocation dynamique)
- == teste l'égalité sur le mot (valeurs immédiate ou pointeur)

GC en Objective Caml

à générations: 2 générations (ancienne et nouvelle)

- `Stop&Copy` sur la nouvelle (GC mineur)
- `Mark&Sweep` incémental sur l'ancienne génération (GC majeur)

caractéristiques:

- Un objet jeune qui survit à 1 GC change de zone.
- Conservation des pointeurs de zone ancienne vers zone jeune
- Si le `Mark&Sweep` échoue, alors un GC compactant est déclenché pour la génération ancienne.

Le module `Gc` permet de contrôler les paramètres du GC.

Module Gc (1)

- statistiques (type *stat* :
  - `Gc.stat : unit -> Gc.stat`
- contrôler les paramètres du tas (type *control*)
  - Gc.get : unit -> Gc.control+
    `Gc.set : Gc.control -> unit`
- forcer le GC :
  - `Gc.minor() : unit -> unit`
  - `Gc.major() : unit -> unit`
  - `Gc.compact() : unit -> unit`

Module Gc (2)

Statistiques:

```
# Gc.stat();;
- : Gc.stat =
{Gc.minor_words = 112065.; promoted_words = 0.;
 major_words = 60074.; minor_collections = 0;
 major_collections = 0; heap_words = 126976;
 heap_chunks = 1; live_words = 60074;
 live_blocks = 11226; free_words = 66902;
 free_blocks = 1; largest_free = 66902;
 fragments = 0; compactions = 0;
 top_heap_words = 126976; stack_size = 53}
```

et fonction d'affichage :

`print_stat : out_channel -> unit`

Module Gc (3)

```
# Gc.stat();;
- : Gc.stat =
{Gc.minor_words = 680793.; promoted_words = 7360.;
 major_words = 117587.; minor_collections = 2;
 major_collections = 4; heap_words = 126976;
 heap_chunks = 1; live_words = 106362;
 live_blocks = 23795; free_words = 20614;
 free_blocks = 1; largest_free = 20614;
 fragments = 0; compactions = 2;
 top_heap_words = 126976; stack_size = 53}
# Gc.major();;
- : unit = ()
# Gc.stat() ;;
- : Gc.stat =
{Gc.minor_words = 702675.; promoted_words = 7360.;
 major_words = 118290.; minor_collections = 2;
 major_collections = 5; heap_words = 126976;
 heap_chunks = 1; live_words = 106671;
 live_blocks = 23880; free_words = 20305;
 free_blocks = 18; largest_free = 19911;
 fragments = 0; compactions = 2;
 top_heap_words = 126976; stack_size = 53}
```

Module Gc (4)

Contrôle:

```
# let c = Gc.get () ;;
val c : Gc.control =
  {Gc.minor_heap_size = 262144;
   major_heap_increment = 126976;
   space_overhead = 80;
   verbose = 0;
   max_overhead = 500;
   stack_limit = 1048576;
   allocation_policy = 0}
```

Module Gc (5)

Par exemple, le champ `verbose` peut prendre des valeurs de 0 à 127 activant 7 indicateurs différents.

```
# c.Gc.verbose <- 127 ;;
- : unit = ()
# Gc.set c ;;
- : unit = ()
# Gc.compact () ;;
```

affichage:

```
Heap compaction requested
<>Sweeping 9223372036854775807 words
Starting new major GC cycle
Marking 9223372036854775807 words
Subphase = 10
Sweeping 9223372036854775807 words
Compacting heap...
done.
```

Les différentes phases du GC sont indiquées ainsi que le nombre d'objets traités.

Mesure de taille de valeurs OCaml (1)

- calcul sur les valeurs vivantes entre 2 Gc encadrant un calcul (si calcul pur)
- exploration d'une valeur en tenant compte du partage

fonction de test : intervalle

```
1  let rec i a b =
2   if  a > b then []
3   else a :: (i (a+1) b);;
```

Mesure de taille de valeurs OCaml (2)

calcul sur les objets vivants :

```
1  let mesure f x =
2    Gc.compact ();
3    let s1 = Gc.stat() in
4      let u = f x in
5      Gc.compact();
6      let s2= Gc.stat() in
7      (u,s1,s2) ;;
8
9  let _,b,c = mesure (i 1) 1000;;
```

Mesure de taille de valeurs OCaml (3)

résultats :

```
#  let _,b,c = mesure (i 1) 1000;;
val b : Gc.stat =
  {Gc.minor_words = 1048056.; promoted_words = 169833.;
   major_words = 330159.; minor_collections = 19; major_collections = 18;
   heap_words = 491520; heap_chunks = 1; live_words = 269812;
   live_blocks = 66474; free_words = 221708; free_blocks = 1;
   largest_free = 221708; fragments = 0; compactions = 9;
   top_heap_words = 491520; stack_size = 73}
val c : Gc.stat =
  {Gc.minor_words = 1051079.; promoted_words = 172856.;
   major_words = 333182.; minor_collections = 21; major_collections = 20;
   heap_words = 491520; heap_chunks = 1; live_words = 272835;
   live_blocks = 67478; free_words = 218685; free_blocks = 1;
   largest_free = 218685; fragments = 0; compactions = 10;
   top_heap_words = 491520; stack_size = 75}
# c.promoted_words -. b.promoted_words;;
- : float = 3023.
# c.live_words - b.live_words;;
- : int = 3023
```

Mesure de taille de valeurs OCaml (4)

créer ou utiliser une fonction d'exploration : bibliothèque `Obj`

```
# Obj.reachable_words;;
- : Obj.t -> int = <fun>
# Obj.repr;;
- : 'a -> Obj.t = <fun>
```

```
# let mesure2 f x =
   let u = f x in
    u,Obj.reachable_words (Obj.repr u) ;;
val mesure2 : ('a -> 'b) -> 'a -> 'b * int = <fun>

# let _,r=mesure2 (i 1) 1000;;
val r : int = 3000
```