TD numéro 3
Garbage collector, partage maximal

Programmation avancée, ENSIIE

Semestre 2, 2013–14

Garbage collection

On considère le programme OCaml suivant :

```
let x =
  let rec y = 1 :: 2 :: 3 :: y in
  4 :: 5 :: 6 :: [] in
let x = List.tl x in
  (* ici *) x
```

1. Représenter l'état de la mémoire au point indiqué par le commentaire `(* ici *)`. Quels sont les parties accessibles depuis le programme ?
2. Appliquer un algorithme de GC de type comptage de références dessus. A-t-on autant libéré la mémoire que possible ?
3. Appliquer un algorithme de type « marquer et nettoyer ».
4. Appliquer un algorithme de GC copiant.

On considère maintenant le programme

```
let a = [1] in
let b = [-1] in
let a = 2 :: a in
let b = -2 :: b in
  ...
let a = n :: a in
let b = −n :: b
```

5. En supposant que les blocs mémoires sont alloués consécutivement à partir d'une mémoire vide, représenter l'état de la mémoire à la fin du programme.
6. Appliquer un algorithme de GC copiant. Que peut-on constater ?

Partage maximal

On considère des arbres binaires d'entiers dont on rappelle le type OCaml :

```
type arbre_binaire =
  Vide
| Noeud of arbre_binaire * int * arbre_binaire
```

Soient les arbres obtenus par le code suivant :

```
let a = Noeud (Noeud (Vide, 1, Vide), 5,
               Noeud(Vide, 3, Noeud (Vide, 1, Vide))) in
let b = Noeud (Noeud(Vide, 3, Noeud (Vide, 1, Vide)), 6,
               Noeud(Vide, 3, Noeud (Vide, 4, Vide)))
```

1. Représenter les arbres `a` et `b` sans partage, puis avec partage maximal.
2. Écrire les constructeurs intelligents qui permettent de faire du partage maximal par hash consing.
3. Représenter le déroulement en mémoire de la construction de `a` et `b`, en supposant qu'on utilise les constructeurs intelligents au lieu de `Noeud` et `Vide`, en incluant la table de hachage servant au hash consing.
4. On suppose que par la suite seul `b` reste accessible, `a` ne l'étant plus. Appliquer un algorithme de GC de type « marquer et nettoyer ».