Master STL - M1
Module CA (4I504)

Compilation Avancée
Documents autorisés

Université Pierre et Marie Curie
Année 2016–2017

Examen du 13 juin 2017

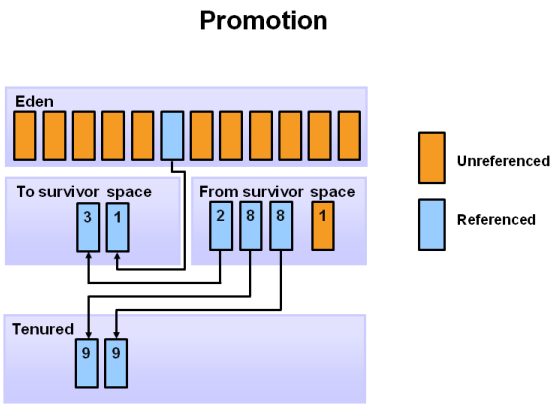
1ère partie (10 points) - à traiter sur une copie séparée

1ère partie : implantation d'un récupérateur automatique de mémoire

On cherche à implanter un GC pour les valeurs allouées en implantant un Stop&Copy sur plusieurs espaces et en datant les valeurs survivantes (c'est-à-dire non désallouées) à la manière d'un des GC de Java. La zone d'allocation dynamique (le tas) est découpée elle-même en deux zones : les valeurs jeunes et les valeurs anciennes. Dans cet exercice on s'intéresse principalement à la zone des valeurs jeunes et à l'implantation d'un GC compactant en utilisant sur cette zone une variante de `Stop&Copy`. Voici les caractéristiques de ce GC :

- Allocation :
  - les nouvelles valeurs sont allouées dans l'espace appelé "Eden" ;
  - quand l'Eden est plein, un GC mineur est déclenché.
- GC mineur
  - il y a deux autres zones pour les valeurs qui survivent à un GC ; on les appelle `fromspace` (ou From survivor space) et `tospace` (ou To survivor space);
  - en dehors des phases de GC la zone `tospace` est toujours complètement libre ;
  - à la fin du GC mineur la zone Eden et la zone `fromspace` sont libérées, et les rôles des deux zones (`tospace` et `fromspace`) sont alors inversés : l'ancienne zone `tospace` est alors considérée la nouvelle zone `fromspace`.
- Datation :
  - chaque valeur survivante voit son âge augmenté de 1 après un GC mineur (cela est effectué pendant la copie) ;
- Promotion
  - les valeurs dépassant un certain âge (par exemple 8) sont déplacées (promues) dans une autre zone du tas, appelée espace des valeurs anciennes (Tenured).

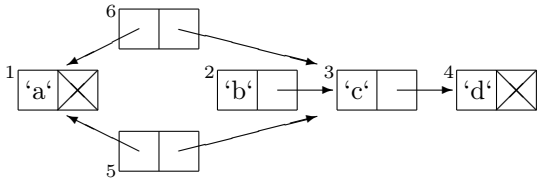
Voici une figure extraite de la documentation Oracle pour Java résumant les différentes étapes lors d'un GC : les flèches indiquent les déplacements des valeurs vivantes de l'Eden et de l'espace `fromspace` des survivants vers l'espace `tospace` des survivants ainsi que la promotion vers la zone des anciennes valeurs (Tenured) de ceux qui ont survécu à 8 GC. A la fin du GC la zone Eden et la zone `fromspace` peuvent être libérées, cette dernière devenant la nouvelle zone `tospace`.



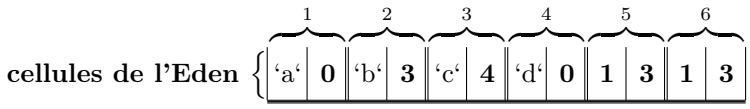
On demande d'écrire cet algorithme dans un pseudo-langage (pseudo-C, pseudo-Caml, pseudo-Java), et de l'illustrer sur le petit exemple suivant :

```
let x = ['a'] ;;
let l  =  List.tl ['b';'c';'d'] ;;
let a  =  match ( x , l )
          with  p  ->  ( fst p , snd p ) ;;
```

Du point de vue allocation, ce code alloue deux listes `['a']` et `['b';'c';'d']` dans le tas, puis une paire `(x,l)`, puis une deuxième paire `(fst p, snd p)` est créée pour relier les deux listes. Au final, on trouve dans l'ensemble des racines la première liste (variable `x`) , la deuxième liste à partir de sa deuxième valeur (valeur `['c'; 'd']` et variable `l`) et la deuxième paire (variable `a`).



Voici ces mêmes allocations avec le tas représenté par un vecteur de cellules de deux cases. Les numéros en haut des cellules indiquent l'ordre de création de ces cellules.



Pour simplifier l'implantation on suppose que le tas ne contient que des cellules regroupant 2 valeurs. Une cellule pourra contenir en plus de ces deux valeurs un entête pour l'âge de la cellule et la nouvelle adresse de la cellule si elle a déjà été déplacée. Cela évite de déplacer de nouveau une cellule déja déplacée. Les valeurs sont soient immédiates (représentées par un caractère), soient allouées (représentées par un entier correspondant à une adresse dans le tas). On suppose qu'il existe un prédicat `est_imm` indiquant si une valeur est immédiate ou non. Une valeur non immédiate correspond à une valeur allouée.

1. Indiquer sur quelles cases de la figure précédente pointent les variables `x`, `l` et `a` de l'exemple, puis faites tourner l'algorithme de déplacement des valeurs survivantes de l'Eden.

2. Définir dans le langage d'implantation de votre choix le type `element` pouvant représenter soit un caractère, soit un entier (pour les adresses), le type `cell` pour les paires d'éléments auxquelles on peut ajouter d'autres champs si nécessaires (pour indiquer l'âge ou l'adresse d'un déplacement), et le type `tas` contenant les différentes zones vues comme des tableaux de `cell`. Ecrire la valeur `montas` de type `tas` et la valeur `ensembleRacines` correspondant respectivement au tas et à l'ensemble des racines de la figure précédente.

3. Ecrire l'algorithme de GC mineur qui à partir de l'ensemble des racines déplace les valeurs survivantes de l'Eden et de la zone `fromspace` vers la zone `tospace` en incrémentant de 1 l'âge des survivants. Pour ceux qui dépassent une certaine limite (ici 8), ils seront déplacés (promus) vers la zone des valeurs anciennes (qui pourra être traitée lors d'un GC majeur). On supposera dans cette question qu'il n'y a pas de débordement de la zone `tospace` pendant un GC, ni que la zone des valeurs anciennes puisse être pleine.

4. On cherche maintenant à traiter les risques de débordement de l'espace `tospace` lors d'un GC mineur. Pour cela on autorise l'augmentation de la taille des espaces des survivants en cours en doublant l'espace `tospace` et on abaissera de 1 l'âge de promotion dans l'espace des valeurs anciennes (avec 2 comme borne inférieure). Selon votre implémentation, il peut être nécessaire de copier les valeurs déjà déplacées dans `tospace` vers le nouveau `tospace` agrandi. Indiquer sur votre copie ce que vous faites des valeurs déjà copiées lors de l'agrandissement de la zone `tospace`.