Département d'informatique
et de recherche opérationnelle
Hiver 2016

Marc Feeley
bureau 3341
feeley@iro.umontreal.ca

Description du projet IFT3065-IFT6232 (Étape 3)
17 mars, 2016

1 Introduction

L'étape 3 du projet consiste à compléter votre compilateur pour qu'il traite un sous-ensemble plus complet du langage de programmation Scheme. La date de remise est le 15 avril. Les objectifs spécifiques sont :

1. Ajouter le type *vecteur* et les fonctions `make-vector`, `vector-length`, `vector-ref`, et `vector-set!`.
2. Ajouter un *ramasse mièttes* (*garbage collector*) de type "stop-and-copy".
3. Faire l'optimisation de *l'appel terminal*.
4. Dans le cas des étudiants en IFT6232, votre compilateur doit être autogène (il doit donc pouvoir se compiler lui-même). Cela demandera entre autres d'implanter les symboles et les fonctions `string->symbol` (pour implanter la lecture de symboles par `read`) et `symbol?` (pour que le compilateur puisse distinguer les symboles des chaînes de caractères).

Notez que toutes les fonctionnalités demandées aux étapes 1 et 2 doivent aussi être opérationnelles. En d'autres termes les trois étapes sont cumulatives.

Pour la notation du projet, une note sera donnée sur l'ensemble des objectifs des étapes 1 à 3 (c'est-à-dire, est-ce que votre compilateur final atteint les objectifs des trois étapes). Si cette note est supérieure à la moyenne des étapes 1 et 2, ce sera la note globale du projet. Sinon, une note sera donnée sur les objectifs spécifiques de la troisième étape et je ferai la moyenne de vos trois notes.

2 Ramasse mièttes

Pour faciliter son écriture, vous devez réaliser le ramasse mièttes en langage C. Évidemment, votre makefile doit être modifié pour inclure la compilation de ce fichier C avec gcc, et si vous utilisez toujours la même logique de compilation dans `compiler.scm`, il faut aussi modifier ce fichier au point où gcc est appelé.

Il faudra que le code généré par votre compilateur pour les allocations mémoire détecte que la zone mémoire du tas qui est libre est insuffisante pour faire l'allocation. Dans ce cas il faut faire un appel à la fonction C qui implante le ramasse mièttes. Notez que la fonction C devra connaître certaines informations importantes comme les pointeurs de début et fin de la pile, de la zone des variables globales, et du tas. Encore une fois pour faciliter l'écriture du ramasse mièttes, ces informations seront contenues dans des variables globales C, et ce sera la responsabilité du code généré par votre compilateur de mettre à jour ces variables.

Voici un squelette de ramasse mièttes pour clarifier :

```
#include <stdio.h>

typedef long word;

/* tenir compte des différences de name mangling */

#ifdef __linux__
#define stack_base _stack_base
#define stack_ptr _stack_ptr
#define gc _gc
#endif

/* variables qui seront mises à jour avant d'appeler gc() */

word *stack_base; /* base de la pile (adresse limite haute) */
word *stack_ptr;  /* valeur de %rsp au moment de l'appel de gc() */

/* le ramasse mièttes */

void gc() {

  printf("*** debut du GC\n");

  printf("stack_base = %p\n", stack_base);
  printf("stack_ptr  = %p\n", stack_ptr);

  /* à compléter! */
}
```

Ici les variables globales `stack_base` et `stack_ptr` sont utilisées par le code généré par le compilateur pour passer les informations de début et fin de la pile à la fonction `gc`. Il faut donc que le code généré par votre compilateur contienne l'instruction :

```
   mov  %rsp, _stack_base(%rip)
```

au début du programme pour que la base de la pile soit mémorisé dans la variable `stack_base`.

En ce qui concerne l'appel de la fonction `gc` il faut utiliser une séquence d'instructions comme celle-ci :

```
mov  %rsp, _stack_ptr(%rip)
mov  %rsp, %rax
and  $-16, %rsp
sub  $8, %rsp
push %rax
call _gc
pop  %rsp
```

Ici la première instruction stocke le pointeur de pile dans la variable `stack_ptr`. Le ramasse mièttes devra traiter tous les mots entre `stack_ptr` et `stack_base` comme des racines (donc assurez vous que votre compilateur stocke uniquement des références d'objets Scheme correctement "taggées" sur la pile, incluant les adresses de retour). Les instructions assembleur suivantes font l'appel de la fonction `gc` après avoir forcé l'alignement de la pile (sur OS X, le `call` d'une fonction C doit obligatoirement se faire lorsque le pointeur de pile est un multiple de 16).

Il faudra ajouter d'autres variables globales pour que le ramasse mièttes sache où commence et se termine le tas (composé du *fromspace* et *tospace*), quelle moitié est présentement le *fromspace*, où commence et se termine la table des variables globales, etc.

3 Rapport

Le travail effectué doit être documenté dans un rapport contenant une description claire de l'ensemble des objectifs visés, la méthodologie adoptée pour atteindre les objectifs, une discussion des problèmes informatiques rencontrés et l'approche utilisée pour les résoudre (algorithmes), une section présentant les résultats de tests unitaires pertinents, et une évaluation honnête de l'atteinte des objectifs (qu'est-ce qui a été accompli et les problèmes restants).

N'hésitez pas à utiliser des diagrammes et choisir quelques bons exemples pour illustrer vos propos dans le rapport. La présentation du rapport doit être sobre (pas de fontes énormes, ni des grosses tables et des diagrammes touffus!). Utilisez un logiciel de traitement de texte (Latex est fortement recommandé!). Je m'attend à des rapports de 7 à 8 pages pour IFT3065 et 8 à 10 pages pour IFT6232 (en police de 12 points, c'est-à-dire 400 à 500 mots par page).

Le rapport doit être en format PDF, et doit se nommer `etape3.pdf` à la racine de votre dépot github.