Interprétation des programmes – TP 2 : Le compilateur OCaml travaille-t-il beaucoup?

Université Paris Diderot – Master 1

(2015-2016)

Cette séance de travaux pratiques a pour objectifs de :
— vous confronter à la différence entre un langage d'ordre supérieur et un langage du premier ordre ;
— vous faire comprendre le *pattern-matching* en le simulant en C.

Exercice 1 (Des fonctions en C) *Donald rentre dans votre bureau très en colère et hurle : « - Comment ça ? On m'a dit que tu programmes en OCaml ! ? Mais enfin, le seul langage qui vaut la peine d'être appris est le C ! Tout s'écrit facilement en C ! » Vous lui répondez calmement que C n'a pas de fonction de première classe mais il répond du tac-au-tac : « Et les pointeurs de fonction ? Ce ne sont pas des fonctions de première classe, peut-être ? »*

Dans cet exercice, nous allons montrer à Donald qu'il se trompe : les pointeurs sur fonctions de C ne sont pas de bonnes représentations pour les fonctions de première classe. Pour cela, nous allons écrire un programme C équivalent au programme OCaml suivant :

```
let range start stop = Array.init (stop - start + 1) (fun i -> i + start)
let stairs = Array.init 10 (range 0)
```

1. *Le programme* `stairs.c` *produit une erreur de segmentation à l'exécution, pouvez-vous expliquer pourquoi ?*
2. *Qu'est-ce qu'une fermeture ? Si vous le savez pas, que comprenez vous de l'article wikipedia suivant ?*

   https://en.wikipedia.org/wiki/Closure_(computer_programming)
3. *Corrigez le programme* `stairs.c` *en représentant les fonctions à l'aide de fermetures à la place des pointeurs.*
4. *Sauriez-vous quand libérer les zones mémoires allouées dynamiquement par votre programme ?*
5. *En conclusion : Est-il théoriquement possible de faire de la programmation fonctionnelle en C ? En pratique, que pensez-vous de l'expressivité de C vu comme un langage fonctionnel ?*

□

Exercice 2 (Analyse de motifs) *Donald rentre de nouveau dans votre bureau et s'écrie : « Je veux bien te croire pour les fonctions de première classe mais ne me dis pas que tu ne pourrais pas te passer de pattern matching ! En C, il y a le switch, c'est exactement pareil ! »*

Dans cet exercice, vous allez simuler en C l'analyse de motifs d'OCaml et toucher du doigt certains aspects subtils de l'implémentation des programmes qui travaillent sur des arbres.

1. *Implémentez le programme* `exp.ml` *en C. Vérifiez sa correction à l'aide d'un script qui s'assure que les entiers affichés sur les sorties standards des deux programmes sont bien les mêmes pour tous les arbres de profondeurs comprises entre 1 et 15.*
2. *À l'aide de la commande* `/usr/bin/time`*, comparez l'efficacité des deux programmes. Avez-vous réussi à écrire un programme C aussi efficace que le programme* OCaml *? Si la réponse est non, optimisez votre programme pour qu'il n'y ait pas plus de 50% d'écart entre les temps de calcul des deux programmes. Pour cela, vous pouvez essayer par exemple (i) de réduire les indirections dans la structure de données d'arbres ; (ii) de maximiser la localité des données allouées dynamiquement ; (iii) d'utiliser des tables de saut pour optimiser les branchements de l'interpréteur ; (iv) de jouer sur les options de compilation de* `gcc`*.*

□