Série d'exercices #5

IFT-2035

May 15, 2025

5.1 Des fonctions pour table

Définir en Haskell des fonctions pour gérer des tables associatives sans utiliser de constructeur de données (tels que `(:)` ou des types algébriques). Plus précisément, définir:

`empty = λx → error "Not found"`	Une table vide
`add x v t`	Ajouter un lien `x` $\mapsto$ `v` à la table `t`
`lookup t x`	Renvoie la valeur `v` liée à `x` dans `t`

De telle manière que:

```
mytab = add "a" 1 (add "b" 2 empty)
lookup mytab "b" ⟹ 2
lookup mytab "c" ⟹ <error: Not Found>
```

Vu que les constructeurs de données ne peuvent pas être utilisés, les tables seront nécessairement représentées par des fonctions/fermetures (i.e. il faudra utiliser des fonctions d'ordre supérieur).

5.2 Structure de données fonctionelle

Soit le type suivant en Haskell qui définit un arbre binaire que l'on peut utiliser pour représenter une table associative (qui associe des *clés* de type `Int` à des valeurs de type β):

```
data TreeMap b = Empty | Node Int b (TreeMap b) (TreeMap b)
```

L'exercice est de définir les opérations typiques sur une telle structure de donnée. Bien sûr, pour être utile l'arbre doit être maintenu dans l'ordre: toutes les clés dans la branche de gauche d'un `Node` doivent être plus petites que la clé du noeud, et vice versa pour la branche de droite.

Il y a trois opérations:

- *tmLookup*: rechercher la valeur associée à une clé passée en paramètre.
- *tmInsert*: ajouter une entrée (donnée sous la forme d'une clé et de sa valeur) dans la table.

- *tmRemove*: enlever une entrée (dont la clé est passée en paramètre).

Ces fonctions doivent être totales (elles terminent toujours et ne doivent jamais signaler d'erreur).

1. Donner un type acceptable pour chacune de ces trois fonctions.
2. Donner le code des deux premières fonctions (points de karma en bonus pour *tmRemove* qui est plus pénible et moins souvent nécessaire).
3. Pour un arbre de profondeur 10 contenant ∼1000 éléments, quel est le coût approximatif de chacune de ces fonctions.

Pour rendre l'exercice plus utile, il est important de faire ces étapes dans l'ordre: i.e. ne pas écrire le code avant d'avoir décidé du type des fonctions.

5.3 Évaluateur

```
type Var = String

-- Expressions du code source en forme ASA.
data Exp = Enum Int               -- Une constante
         | Evar Var               -- Une variable
         | Elambda Var Exp        -- Une fonction
         | Ecall Exp Exp          -- Un appel de fonction

-- Valeurs renvoyées.
data Val = Vnum Int               -- Un nombre entier
         | Vfun Var Exp           -- Une fonction

elookup x ((x1,v1):env) =
    if x == x1 then v1 else elookup x env

eval env (Enum n) = Vnum n
eval env (Evar x) = eval (elookup x env)
eval env (Elambda arg body) = Vfun arg body
eval env (Ecall fun actual) =
    case eval env fun of
      Vfun formal body -> eval ((formal,actual):env) body
```

Soit les déclarations ci-dessus utilisées pour un interpréteur:

1. Donner le type des deux fonctions. Vérifier que le code est typé correctement et corriger les éventuelles erreurs.
2. Cet évaluateur implante-t-il l'appel par valeur ou par nom?
   Indiquer quelle partie du code vous indique clairement la réponse.
   Le changer pour qu'il implante l'autre genre de passage d'arguments.
3. Cet évaluateur implante-t-il la portée lexicale ou dynamique?
   Indiquer quelle partie du code vous indique clairement la réponse.
   Le changer pour qu'il implante l'autre genre de portée.

Points bonus:

- Discuter de l'influence de l'ordre d'évaluation de Haskell sur vos réponses aux points 2 et 3.
- Comment faut-il changer ce code pour que l'évaluateur hérite la règle de portée utilisée par le méta langage (Haskell). En d'autres termes, que votre évaluateur implante la portée dynamique si Haskell utilisait la portée dynamique, et qu'il implante la portée statique si Haskell utilisait la portée statique.
- Entre le code fourni, celui obtenu après le point 2, et celui obtenu après le point 3, vous avez 3 variantes parmi les 4 combinaisons possibles. Comment faut-il changer le code pour obtenir la combinaison manquante?