Série d'exercices #6

IFT-2035

October 10, 2023

6.1 Ordre et Portée

Définir dans un langage fonctionnel hypothétique une fonction qui renvoie:

- 0 si le langage utilisé obéi la portée statique et l'appel par valeur
- 1 si le langage utilisé obéi la portée statique et l'appel par nom
- 2 si le langage utilisé obéi la portée dynamique et l'appel par valeur
- 3 si le langage utilisé obéi la portée dynamique et l'appel par nom

6.2 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?

6.3 Structure de données infinie

Les listes infinies sont souvent appelées *streams*. En Haskell, l'ordre d'évaluation utilisé permet d'utiliser n'importe quelle structure de donnée infinie sans effort particulier. Prenons par exemple les définitions ci-dessous:

```
zeros = 0 : zeros
uns = 1 : uns
numbers = 0 : zipWith (+) uns numbers
f = 0 : 1 : zipWith (+) f (tail f)
```

De plus, Haskell prédéfini les opérations suivantes:

```
(x:_)  !! 0 = x
(_:xs) !! n = xs !! (n - 1)

take 0 _ = []
take _ [] = []
take n (x:xs) = x : take (n - 1) xs
```

Avec les définitions ci-dessus, nous avons:

$$\begin{array}{lll} \textit{take}\ 5\ \textit{uns} & \rightsquigarrow^* & [1,1,1,1,1] \\ \textit{take}\ 5\ \textit{numbers} & \rightsquigarrow^* & [0,1,2,3,4] \\ \textit{numbers}\ !!\ 3 & \rightsquigarrow^* & 3 \end{array}$$

1. Montrer les étapes de l'évaluation de:

```
f !! 3
```

Expliquer en détail la notation que vous avez choisi d'utiliser. Attention à l'utiliser de manière cohérente et rigoureuse.

2. Soit la liste infinie de nombres suivante:

```
(1 1/2 1/6 1/24 1/120 ... 1/n! ...
```

Définir cette liste récursivement en utilisant *zipWith*, la liste infinie des entiers *numbers* et la liste circulaire `uns`.