Série d'exercices #4

IFT-2035

September 23, 2024

4.1 Renommage α

Soit le code ci-dessous qui est écrit en Haskell et utilise donc la portée lexicale:

```
λx → λy →
let f = λx → x + 2 in
let g x = λg → f (g x) in
let g (x, f) = f x
in λf → g (x, f)
```

Renommer toutes les variables (en ajoutant un 0, 1, 2, ... aux identifiants) pour que chaque variable ait un nom différent des autres. Bien sûr ce renommage ne doit pas changer la sémantique du code.

4.2 Portée et passage d'arguments

Soit le code ci-dessous dans un langage hypothétique Shmaskell (qui utilise la même syntaxe que Haskell, mais qui n'est pas pur):

$$
\begin{array}{l}
\mathsf{let}\ x = 10\ \mathsf{in} \\
\mathsf{let}\ get2x = \lambda\_ \to (print\ \text{“}getting2x''; x * 2)\ \mathsf{in} \\
\mathsf{let}\ f = \lambda x \to \lambda y \to x + get2x() + x + y\ \mathsf{in} \\
f\ (get2x())\ 7
\end{array}
$$

Donner la valeur de retour ainsi que le(s) message(s) imprimé(s) selon que Shmaskell utilise:

- Portée statique et passage d'arguments par valeur.
- Portée statique et passage d'arguments par nom.
- Portée dynamique et passage d'arguments par valeur.
- Portée dynamique et passage d'arguments par nom.

4.3 Mini-évaluateur en manque de fonctions

```
type Var = String

-- Expressions du code source en forme ASA.
data Exp = Enum Int           -- Une constante
         | Evar Var           -- Une variable
         | Elet Var Exp Exp   -- Une expr "let x = e1 in e2"
         | Ecall Exp Exp      -- Un appel de fonction

-- Valeurs renvoyées.
data Val = Vnum Int             -- Un nombre entier
         | Vprim (Val → Val)    -- Une primitive

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

eval env (Enum n) = Vnum n
eval env (Evar x) = elookup env x
eval env (Elet x e1 e2) =
   let v = eval env e1 in eval ((x,v):env) e2
eval env (Ecall fun actual) =
   case eval env fun of
     Vprim f -> f (eval actual)
```

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. Étendre ce petit langage avec la possibilité de définir de nouvelles fonctions. I.e. Ajouter un constructeur d'expression `Elambda`, un constructeur de valeur correspondante `Vlambda`, ainsi que les cas pour gérer ces constructeurs dans `eval`.

4.4 Ordre et Portée

Définir dans un langage fonctionnel non-pur 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

4.5 Des trous typés

Dans le code ci-dessous, $\bullet$ représente une *expression* manquante. Donner le type de l'expression manquante. E.g. pour la question 0, la réponse pourrait être: $\bullet : \mathsf{Int} \to \alpha$. Comme d'habitude, vous pouvez présumer que toutes les entités numériques sont de type Int.

0. $\bullet\ 1$
1. $\lambda x \to (2 + x - \bullet)$
2. $[[10, 9, 8], \bullet]$
3. $([(+), (-)], \bullet)$
4. $[(8, 3), \bullet]$
5. $\lambda x \to (x + \bullet\ x)$
6. $\lambda x \to (\bullet\ (x + 1)\ (x - 1), x)$
7. $\lambda x \to \lambda y \to (x\ y + \bullet\ x)$
8. $map\ (\lambda x \to x + 1)\ \bullet$
9. $map\ \bullet\ [5, 6, 7]$
10. $\mathsf{let}\ x = \bullet\ \mathsf{in}\ map\ snd\ (x\ [42])$

Attention à ne pas donner de type trop spécifique (e.g. $\mathsf{Int} \to \mathsf{Int}$ pour 0) ni trop générique (e.g. $\alpha \to \beta$ pour 0).

Deuxième tour: Pour chaque $\bullet$, donner un exemple de code qui a le type que vous avez spécifié. De nouveau, assurez-vous que ce n'est pas simplement un morceau de code qui aurait le droit de remplacer $\bullet$, mais bien un morceau de code qui a le type du trou.

Rappel: les fonctions *map* et *snd* sont (pré)définies comme suit:

$$map\ f\ [] = []$$
$$map\ f\ (x : xs) = f\ x : map\ f\ xs$$
$$snd\ (x, y) = y$$