Série d'exercices #4

IFT-2035

May 12, 2025

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 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. let $x = \bullet$ in *map snd* $(x\ [42])$

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

Deuxième tour: Pour chaque •, 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 •, mais bien un morceau de code qui a le type du trou.

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

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

4.3 Un petit évaluateur

Soit les déclarations suivantes pour un mini-langage d'expressions:

```
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
```

Les fonctions prédéfinies sont les quatre opérations arithmétiques, liées aux variables "+", "-", "*", et "/", respectivement. Ces fonctions prennent deux arguments qui sont passés de manière currifiée. Par exemple une expression telle que "let $x = 3$ in $x + 4$" est représentée par la structure suivante de type *Exp*:

```
sampleExp = Elet "x" (Enum 3)
                 (Ecall (Ecall (Evar "+") (Evar "x")) (Enum 4))
```

L'environnement initial `env0` prédéfini les quatre fonctions:

```
mkPrim :: (Int → Int → Int) → Val
mkPrim f = Vprim (λ(Vnum x) → Vprim (λ(Vnum y) → Vnum (f x y)))

-- L'environnement initial qui contient toutes les primitives.
type Env = [(Var, Val)]
env0 :: Env
env0 = [("+", mkPrim (+)), ("-", mkPrim (-)),
        ("*", mkPrim (*)), ("/", mkPrim div)]
```

Écrire la fonction *eval* qui prend un environnement qui décrit les variables liées (et leur valeur) ainsi qu'une expression et qui renvoie le résultat de l'évaluation de l'expression. I.e.:

```
eval :: Env → Exp → Val
```

de sorte que `eval env0 sampleExp` renvoie `Vnum 7`.