Série d'exercices #5

IFT-2035

September 30, 2024

5.1 Des expressions typées

Donner le type des expressions Haskell ci-dessous. Par exemple, pour la question 0, la réponse pourrait être: (Int, Int) .

Le type donné devrait être aussi polymorphe que possible. Cependant, vous pouvez présumer que les opérations numériques n'opèrent que sur des objets de type *Int*, plutôt que d'utiliser des *classes de types*.

0. $(2, 3)$
1. $["a", 3]$
2. $[(1, 2), (3, 4)]$
3. $\lambda x \to x : []$
4. $map\ (\lambda x \to x)\ (1, 2, 3)$
5. $[\lambda x \to x, \lambda y \to y + 1]$
6. $\lambda x\ y \to x + y$
7. $\mathsf{let}\ f\ x = x\ (x + 1)\ \mathsf{in}\ f$
8. $fst \circ fst$
9. $\mathsf{let}\ f\ x = f\ (x + 1)\ \mathsf{in}\ f$
10. $\mathsf{let}\ f\ x\ y = (x + 1)\ \mathsf{in}\ f$
11. $map\ (\lambda x \to x + 1)$

Précisions: Les fonctions *map*, *fst*, et $\circ$ sont (pré)définies comme suit:

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

$fst\ (x, y) = x$
$f_1 \circ f_2 = \lambda x \to f_1\ (f_2\ x)$

5.2 Équivalence

Deux expressions ne sont équivalentes que si l'on peut remplacer l'une par l'autre dans n'importe quel programme sans affecter le comportement de ce programme (les différences de performance ne comptent pas).
E.g. `x + y` est équivalent à `y + x`, mais `x + y - y` n'est pas équivalent à `x` puisqu'il se peut que `y` n'existe pas, n'aie pas le bon type, ou que son évaluation ne termine pas.

Indiquer si les expressions Haskell ci-dessous sont équivalentes ou pas.

1. $\mathsf{let}\ f\ x\ y = x + y\ \mathsf{in}\ f\ a\ b \stackrel{?}{=} \mathsf{let}\ f\ (x, y) = x + y\ \mathsf{in}\ f\ (a, b)$
2. $\mathsf{let}\ f\ x = x + y\ \mathsf{in}\ g\ f \stackrel{?}{=} \mathsf{let}\ f = \lambda x \to x + y\ \mathsf{in}\ g\ f$
3. $\mathsf{let}\ x = 3\ \mathsf{in}\ g\ x \stackrel{?}{=} g\ 3$
4. $\lambda x \to x + y \stackrel{?}{=} \lambda y \to y + x$
5. $\lambda x\ y \to g\ (x + y) \stackrel{?}{=} \lambda a\ b \to g\ (b + a)$
6. $\lambda x\ x \to g\ x \stackrel{?}{=} \lambda a\ b \to g\ a$
7. $\lambda x\ y \to g\ y\ x \stackrel{?}{=} \lambda(x, y) \to g\ (y, x)$
8. $\lambda a \to [a, a] \stackrel{?}{=} \lambda b \to (b : b)$
9. $\lambda a \to \lambda b \to b\ a \stackrel{?}{=} \lambda y\ x \to x\ y$
10. $map\ (\lambda a \to a\ g)\ [b] \stackrel{?}{=} [b\ g]$

5.3 Des fonctions pour table

Définir en Haskell des fonctions pour gérer des tables associatives sans utiliser de constructeur (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>
```

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).

5.4 Programmer sans variables

La programmation *point-free* ou *sans variable* consiste en un style de programmation fonctionnelle où l'on combine des fonctions pour en construire d'autres plutôt que d'utiliser la forme λ[1]. Les fonctions de base sont généralement appelées des *combinateurs*. En utilisant les fonctions prédéfinies telles que `(.)` (composition de fonctions, habituellement notée $\circ$), `(==)`, `(/)`, `not` ainsi que les fonctions ci-dessous:

```
flip f x y = f y x
map2 f (x:xs, y:ys) = f x y :  map2 f (xs, ys)
map2 f _ = []
```

mais sans utiliser λ (ni un λ implicite caché dans du sucre syntaxique, bien sûr), définir les fonctions suivantes:

`pas_zero 2` $\Longrightarrow$ `True` `pas_zero 0` $\Longrightarrow$ `False`
`moitie 13.2` $\Longrightarrow$ `6.6`
`somme [1,4,8]` $\Longrightarrow$ `13`
`produit_scalaire ([1,2,3], [2,3,4])` $\Longrightarrow$ `20`

Que vaut: `flip (.)  moitie sqrt 18` ?

[1] Le nom *point-free* vient du fait que les variables sont traditionnellement introduites dans le λ-calcul par des expressions qui utilisent une syntaxe où la variable est suivie d'un point, telles que $\lambda x.e$, $\forall x.e$, $\exists x.e$, ou $\mu x.e$