Examen Intra

IFT-2035

May 22, 2015

Directives

- Documentation autorisée: aucune.
- Répondre sur le questionnaire dans l'espace libre qui suit chaque question. Utiliser le verso de la page si nécessaire.
- Chaque question vaut 5 points pour un total maximum de 25 points.
- Les questions ne sont pas placées par ordre de difficulté.

0 Nom et prénom (1 point de bonus)

Écrire son nom et prénom et son code permanent en haut de chaque page.

1 Syntaxe

Soient les expressions suivantes en notation préfixe:

1. $+ \ * \ - \ a \ b \ c \ d$
2. $* \ a \ - \ b \ + \ c \ d$
3. $\wedge \ a \ \wedge \ + \ b \ c \ d$
4. $< \ \mathsf{sqrt} \ + \ a \ b \ - \ c \ d$
5. $= \ \mathsf{not} \ a \ \&\& \ < \ b \ \mathsf{sin} \ c \ \mathsf{empty} \ d$

Après s'être souvenu que la mise à la puissance (représentée par $\wedge$) est associative à droite, écrire ces expressions en format infixe, postfixe et en arbre de syntaxe abstraite. En notation infixe utiliser la notation a^b pour la mise à la puissance et utiliser un minimum de parenthèses.

2 Types

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.

0. $(2, 3)$
1. $[\text{"}a\text{"}, 3]$
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:

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

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

3 Égalité

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

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. $map\ (\lambda a \to a\ g)\ [b] \stackrel{?}{=} [b\ g]$

4 Portée dynamique

Soit le code ci-dessous qui est écrit dans un langage hypothétique Haskedd qui est identique à Haskell sauf qu'il utilise la portée dynamique:

```
let p = "h" in
let mr x = (x, p) in
let f1 x = (let p = "o" in mr (x+1)) in
let f2 l =
    let p = "p" in
    [mr 1] ++ (map f1 l) ++ [mr 9]
in [mr 0] ++ f2 [5, 6, 7]
```

1. Calculer la valeur renvoyée par ce code.
2. Calculer la valeur que renverrait ce code si on le passait par erreur à un interpréteur Haskell (où la portée est lexicale).
3. Réécrire ce code en Haskell. I.e. Écrire une version équivalente pour un langage où la portée est lexicale, tout en restant aussi fidèle que possible au code original.
4. Donner la définition en Haskedd du *map* utilisé ci-dessus.

5 Structures de données

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 [Bool] à des valeurs de type *b*):

```
data Maybe a   = Nothing | Just a
data TreeMap b = Empty | Node (TreeMap b) (Maybe b) (TreeMap b)
```

L'opération de recherche a le type suivant:

```
tmLookup :: [Bool] -> TreeMap b -> b
```

et fonctionne sur le principe de lire la liste de booléens comme une séquence de commandes où *True* indique de descendre dans le sous-arbre de gauche et *False* indique de descendre dans le sous-arbre de droite.

1. Écrire la fonction *tmLookup* ci-dessus.
2. Proposer un autre type pour *tmLookup* qui permettrait une implantation plus robuste.
3. Donner le type possible de la fonction *tmInsert* qui permet d'insérer un élément dans une table.
4. De même pour *tmRemove* qui permet d'enlever un élément de la table et qui renvoie la valeur qu'avait cet élément (si applicable).
5. Écrire la fonction *tmRemove*.