Série d'exercices #3

IFT-2035

May 9, 2023

3.1 Micro optimizer

Soit le code Haskell ci-dessous:

```
data Exp = Enum Int          -- Une constante
         | Evar String       -- Une référence à une variable
         | Eplus Exp Exp     -- e1 + e2
         | Etimes Exp Exp    -- e1 * e2

optimize ::  Exp → Exp
```

Exp est un type Haskell (utilisé ici comme un métalangage) qui représente des expressions simples d'un *langage objet* incluant uniquement des opérateurs arithmétiques. Écrire la fonction *optimize* qui devrait simplifier une expression en éliminant *toutes* les multiplications par 1 et 0, ainsi que les additions à 0.

3.2 Mini Évaluateur

Soit le code Haskell ci-dessous:

```
data Exp = Enum Int          -- Une constante
         | Eplus Exp Exp     -- e1 + e2
         | Etimes Exp Exp    -- e1 * e2
         | Eneg Exp          -- (- e)
         | Egt Exp Exp       -- e1 > e2
         | Enot Exp          -- (not e)
         | Eif Exp Exp Exp   -- if e1 then e2 else e3

data Val = Vnum Int
         | Vbool Bool

eval ::  Exp → Val
```

Exp est un type Haskell (utilisé ici comme un métalangage) qui représente des expressions simples d'un *langage objet* incluant des opérateurs arithmétiques et booléens. *Val* est un type qui représente des valeurs entières ou booléennes. *eval* est une function qui doit évaluer une expression et retourner la valeur qui en résulte.

1. Écrire la fonction *eval*.

2. Proposer des changements au type `Exp` qui permettraient de réduire la redondance dans votre code.

3. Proposer des changements pour ajouter la notion de *variable* dans le langage objet.

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