Fiche de TD no. 5

Le λ-calcul non typé

Exercice 1 : *Arbre abstrait, redex.* Dessinez l'arbre abstrait de chacun des λ-termes suivants et, ensuite, trouvez tous les redexes dans un tel terme :

1. $(\lambda x.\lambda y.((\lambda z.(z\ z))\ x))\ \hat{1}\ \texttt{SUCC}$;
2. $\texttt{SUCC}\ (\texttt{SUCC}\ \hat{1})$;
3. $\texttt{FIX}\ \texttt{SUCC}$;

où :

$$\hat{1} := \lambda f.\lambda x.(f\ x)\,, \qquad \texttt{SUCC} := \lambda n.\lambda f.\lambda x.f\ (n\ f\ x)\,, \qquad \texttt{FIX} := \lambda g.(\lambda x.g\,(x\,x))\,(\lambda x.g\,(x\,x))\,.$$

Exercice 2. Considérez les termes suivants du λ-calcul :

$$\texttt{TRUE} := \lambda x.\lambda y.x\,, \qquad \texttt{FALSE} := \lambda x.\lambda y.y\,, \qquad \texttt{IFTHENELSE} := \lambda p.\lambda a.\lambda b.pab\,.$$

Soient t_1, t_2 deux termes arbitraires ; montrez que

$$\texttt{IFTHENELSE}\ \texttt{TRUE}\ t_1\ t_2\ \xrightarrow{*}_\beta\ t_1 \quad \text{et} \quad \texttt{IFTHENELSE}\ \texttt{FALSE}\ t_1\ t_2\ \xrightarrow{*}_\beta\ t_2\,.$$

Exercice 3. Considérez le terme suivant du λ-calcul :

$$\texttt{FIX} := \lambda g.(\lambda x.g(xx))(\lambda x.g(xx))$$

Soit t un terme arbitraire ; montrez qu'il existe un terme t' tel que

$$\texttt{FIX}\ t\ \xrightarrow{*}_\beta t' \quad \text{et} \quad \texttt{FIX}\ t\ \xrightarrow{*}_\beta\ t\ t'\,.$$

Ainsi—si on déclare deux termes t_1, t_2 équivalents chaque fois que $t_i \xrightarrow{*}_\beta t'$, pour $i = 1, 2$ et un quelque terme t'—on a l'équivalence

$$\texttt{FIX}\ t \equiv t\ (\texttt{FIX}\ t)$$

et on pourra considérer $\texttt{FIX}\ t$ comme le point fixe de la fonction t.

Exercice 4 : *Stratégies d'évaluation.* Utilisez les différentes stratégies—stratégie par l'intérieur (appelée aussi *ordre applicatif*), stratégie par l'extérieur (appelée aussi *ordre normal*), stratégie « *call by value* » et stratégie « *call by name* »— pour évaluer le premier termes proposé à l'exercice 1. S'il vous reste du temps, évaluez (avec les quatre stratégies) aussi le deuxième et troisième terme de cet exercice.

Évaluation, en Haskell

Exercice 5. Utilisez d'abord la stratégie *call-by-value* et ensuite la stratégie *call-by-name* pour évaluer les expressions suivantes :

```
(\b -> \x -> \y -> if b then x else y) (True && True) [] (replicate 2 '*')
take 2 (repeat '*')
add (Succ Zero) Zero
(\x -> \y -> x) 5 omega where omega = omega + 1
```

Exercice 6. Considérez les définitions suivantes des la fonction `fib :: Int -> Int` qui associe à un entier n le n-ème nombre de Fibonacci :

```
fib1,fib2,fib3 :: Int -> Int

fib1 0 = 1
fib1 1 = 1
fib1 n = fib1 (n-1) + fib1 (n-2)

fib2 0 = 1
fib2 1 = 1
fib2 n = fibliste !! (n -1) + fibliste !! (n-2)
fibliste = map fib2 [0..]

fib3 n = fib3acc n 1 1
  where
    fib3acc 0 n m = n
    fib3acc k n m =  fib3acc (k -1) m (m + n)
```

Quelle est, à votre avis, la plus (resp. la moins) performante ? Justifiez votre réponse.

Exercice 7. Démontrez que les fonctions `fib1` et `fib3` depuis l'exercice 6 calculent la même suite infinie de nombres.