

## Fiche de TD no. 5

### Le $\lambda$ -calcul non typé

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

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

où :

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

**Exercice 2.** Considérez les termes suivants du  $\lambda$ -calcul :

$$\text{TRUE} := \lambda x.\lambda y.x, \quad \text{FALSE} := \lambda x.\lambda y.y, \quad \text{IFTHENELSE} := \lambda p.\lambda a.\lambda b.p a b.$$

Soient  $t_1, t_2$  deux termes arbitraires ; montrez que

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

On pourra supposer que  $t_1$  et  $t_2$  ne contiennent pas les variables  $x, y, a, b, p$ , afin d'éviter le renommage des variables lors des substitutions.

**Solution.** On traite le premier cas.

$$\begin{aligned} \text{IFTHENELSE TRUE } t_1 t_2 &= (\lambda p.\lambda a.\lambda b.p a b) (\lambda x.\lambda y.x) t_1 t_2 \\ &\rightarrow_{\beta} \lambda a.\lambda b.((\lambda x.\lambda y.x) a b) t_1 t_2 \\ &\rightarrow_{\beta} \lambda b.((\lambda x.\lambda y.x) t_1 b) t_1 t_2 \\ &\rightarrow_{\beta} (\lambda x.\lambda y.x) t_1 t_2 \\ &\rightarrow_{\beta} (\lambda y.t_1) t_2 \\ &\rightarrow_{\beta} t_1[t_2/y] = t_1, \text{ car } y \text{ n'apparaît pas dans } t_1. \end{aligned}$$

□

**Exercice 3.** Considérez le terme suivant du  $\lambda$ -calcul :

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

Soit  $t$  un terme arbitraire (on pourra supposer que  $t$  ne contient pas les variables  $x$  et  $g$ ) ; montrez qu'il existe un terme  $t'$  tel que

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

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

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

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

**Solution.** Nous avons :

$$\begin{aligned} \text{FIX } t &= \lambda g.((\lambda x.g(x x)) (\lambda x.g(x x))) t \\ &\rightarrow_{\beta} (\lambda x.t(x x))(\lambda x.t(x x)) \\ &\rightarrow_{\beta} t((\lambda x.t(x x)) (\lambda x.t(x x))). \end{aligned}$$

Ainsi, si on pose

$$t' := (\lambda x.t(x x)) (\lambda x.t(x x)),$$

on a alors :

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

Remarquez que nous avons utilisé l'évaluation par l'extérieur; que se passe-t'il si on utilise la stratégie d'évaluation par l'intérieur?  $\square$

**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.

**Solution.**

1-Evaluation par l'intérieur :

$$\begin{aligned} (\lambda x.\lambda y.((\lambda z.(z z)) x)) \hat{\text{I}} \text{SUCC} &= (\lambda x.\lambda y.((\lambda z.(z z)) x)) (\lambda f.\lambda x.f x) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\rightarrow_{\beta} (\lambda x.\lambda y.(x x)) (\lambda f.\lambda x.f x) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\rightarrow_{\beta} \lambda y.((\lambda f.\lambda x.f x) (\lambda f.\lambda x.f x)) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\rightarrow_{\beta} \lambda y.(\lambda x.((\lambda f.\lambda x.f x) x)) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\sim_{\alpha} \lambda y.(\lambda x.((\lambda f.\lambda z.(f z)) x)) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\rightarrow_{\beta} \lambda y.(\lambda x.(\lambda z.(x z))) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\rightarrow_{\beta} \lambda x.(\lambda z.(x z)) \\ &\sim_{\alpha} \lambda x.(\lambda y.(x y)). \end{aligned}$$

2-Évaluation par l'extérieur :

$$\begin{aligned} (\lambda x.\lambda y.((\lambda z.(z z)) x)) \hat{\text{I}} \text{SUCC} &= (\lambda x.\lambda y.((\lambda z.(z z)) x)) (\lambda f.\lambda x.(f x)) (\lambda n.\lambda f.\lambda x.(f(n f x))) \\ &\rightarrow_{\beta} (\lambda y.((\lambda z.(z z)) (\lambda f.\lambda x.(f x)))) \lambda n.\lambda f.\lambda x.(f(n f x)) \\ &\rightarrow_{\beta} (\lambda z.(z z))(\lambda f.\lambda x.(f x)) \\ &\rightarrow_{\beta} (\lambda f.\lambda x.f x) (\lambda f.\lambda x.f x) \\ &\rightarrow_{\beta} \lambda x.((\lambda f.\lambda x.f x) x) \\ &\sim_{\alpha} \lambda x.((\lambda f.\lambda y.(f y)) x) \\ &\rightarrow_{\beta} \lambda x.(\lambda y.(x y)). \end{aligned}$$

3 et 4-Stratégie Call-by-value (comme l'évaluation par l'intérieur, mais on ne réduit pas sous un  $\lambda$ ) et stratégie Call-by-name (comme l'évaluation par l'extérieur, mais on ne réduit pas sous un  $\lambda$ ). La contrainte de ne pas réduire sous un  $\lambda$  implique, dans notre cas, que les deux stratégies coïncident. Nous utiliserons les

noms  $\hat{1}$  et `SUCC` pour lisibilité.

$$\begin{aligned}
 (\lambda x. \lambda y. ((\lambda z. (z z)) x)) \hat{1} \text{SUCC} &\rightarrow_{\beta} (\lambda y. ((\lambda z. (z z)) \hat{1})) \text{SUCC} \\
 &\rightarrow_{\beta} (\lambda z. (z z)) \hat{1} \\
 &\rightarrow_{\beta} \hat{1} \hat{1} \\
 &= (\lambda f. \lambda x. f x) (\lambda f. \lambda x. f x) \\
 &\rightarrow_{\beta} \lambda x. ((\lambda f. \lambda x. f x) x).
 \end{aligned}$$

□

## 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.