LIFAP5 – Programmation fonctionnelle pour le WEB
TD1 – introduction au λ-calcul et à JavaScript

Licence informatique UCBL – Printemps 2020–2021

Exercice 1 : Typage et évaluation en λ-calcul

1. Pour chacune des λ-expressions suivantes, donner son type en supposant que le seul type primitif est `number` puis l'évaluer *en explicitant chaque β-réduction*. Si l'expression n'est pas typable, expliquer pourquoi. On supposera que l'addition notée $+$ a pour type `number` $\rightarrow$ `number` $\rightarrow$ `number` et se réduit par calcul arithmétique usuel, de même que pour la multiplication notée $*$. Par exemple, $(\lambda x.x + 3)\ 2$ a pour type `number` et peut se réduire en $(\lambda x.x + 3)\ 2 \overset{x}{\rightsquigarrow} 2 + 3 \rightsquigarrow 5$
   1. $(\lambda x.\lambda y.(x + y))\ 3\ 4$
   2. $(\lambda x.\lambda y.(x\ y))\ 3\ 4$
   3. $\lambda x.(x\ x)$
   4. $(\lambda x.\lambda y.(y\ (x + 2)))\ 5\ (\lambda z.(z * 3))$
   5. $(\lambda x.((\lambda y.(x + y))\ x))\ 5$
   6. $(\lambda x.((x\ (\lambda y.(y + 2))) + (x\ (\lambda z.(z * 3)))))\quad ((\lambda u.\lambda w.(w\ u))\ 5)$
   7. $(\lambda x.\lambda y.(x\ y))(\lambda z.z)$
   8. $\lambda x.(x + 3)(\lambda y.y)$
   9. $(\lambda x.\lambda y.x)$
   10. $(\lambda x.\lambda y.x)\ 3$
   11. $(\lambda x.\lambda y.(y\ (3 + x)))\ 4\ (\lambda z.(z * 2))$
   12. $(\lambda x.((\lambda y.(y * x))\ x))\ 5$
2. Donner deux séquences de réductions *différentes* de l'expression $(\lambda x.x + 5)((\lambda z.z)\ 2)$

Exercice 2 : Du λ-calcul au JavaScript et vice-versa

1. Réécrire les expressions suivantes du λ-calcul en JavaScript en utilisant la notation `=>` :
   — $\lambda f.\lambda x.f(f\ x)$
   — $\lambda x.\lambda y.(x\ y)\ y$
   — $\mathbf{I} = \lambda x.x$
   — $\mathbf{K} = \lambda x.\lambda y.x$
   — $\mathbf{S} = \lambda x.\lambda y.\lambda z.(x\ z)(y\ z)$
2. Montrer que $\mathbf{SKK}x$ et $\mathbf{I}x$ se β-réduisent en le même terme. Conclure sur $\mathbf{I}$.
3. Réécrire les fonctions JavaScript suivantes en λ-calcul :
   — `let fn1 = f => g => x => f(g(x))`
   — `let fn2 = f => g => x => g(x)+ f(x)`
4. Soient les fonctions JavaScript suivantes `let f0 = x => 2 + x` et `let f1 = x => 2 * x`. Calculer `fn1(f0)(f1)(3)`.

5. Réécrire en JavaScript une version β-réduite (plus simple) de `fn1(f0)(f1)` avec les fonctions `f0` et `f1` de la question précédente.
6. Soit la fonction `let fn3 = x => x(x)`. Donner le résultat ou le message d'erreur qu'affiche la console JavaScript lors de l'exécution des expressions suivantes :
   — `fn3(3)`
   — `fn3(x => x)`
   — `fn3(x => x)(3)`
   — `fn3(fn3(x => x))(3)`
7. Considérons la fonction JavaScript `let w = x => x(x)`. L'exécution de `w(x => x +1)` produit le résultat `"x => x +11"`. Expliquez.

Exercice 3 : Codage dans le λ-calcul

Dans cet exercice on va s'intéresser au codage des booléens en λ-calcul pur appelé *booléens de Church*. Dans ce codage, les valeurs de vérités sont *des fonctions* et les fonctions booléennes des *fonctions d'ordre supérieur*. On définit les termes suivants qui représentent « vrai », « faux » et la fonction « et » :

— $\mathbf{T} = \lambda x.\lambda y.x$
— $\mathbf{F} = \lambda x.\lambda y.y$
— $\mathbf{AND} = \lambda x.\lambda y.((x\ y)\ \mathbf{F})$, soit $\lambda x.\lambda y.x\ y\ \mathbf{F}$ avec les parenthèses implicites

1. Vérifier les réductions suivantes

$$\begin{aligned} \mathbf{AND}(\mathbf{T})(\mathbf{T}) &\rightsquigarrow \mathbf{T} \\ \mathbf{AND}(\mathbf{T})(\mathbf{F}) &\rightsquigarrow \mathbf{F} \\ \mathbf{AND}(\mathbf{F})(\mathbf{T}) &\rightsquigarrow \mathbf{F} \\ \mathbf{AND}(\mathbf{F})(\mathbf{F}) &\rightsquigarrow \mathbf{F} \end{aligned}$$

2. Quelle est la fonction booléenne associée à $\mathbf{XXX} = \lambda x.\lambda y.x\ y\ x$?
3. Qu'est ce qui se passe quand on évalue $\mathbf{AND}\ x\ y$ où x ou y n'est *pas* la représentation d'un booléen ?
4. Donner une représentation de la fonction booléenne « non ».
5. Donner une représentation de la fonction booléenne « ou ».
6. Montrer que le terme $\mathbf{IF} = \lambda x.x$ permet de représenter l'instruction conditionnelle `IF cond THEN M ELSE N`.
7. Coder la représentation des booléens en JavaScript et écrire une fonction qui va afficher les tables de vérité des opérations binaires définies précédemment
8. Écrire une fonction JavaScript qui va comparer si deux opérations codées en λ-calcul ont la même table de vérité.