LIFAP5 – Programmation fonctionnelle pour le WEB
TD2 – programmation fonctionnelle en JavaScript

Licence informatique UCBL – Printemps 2021–2022

Exercice 1 : Curryfication et application partielle

Dans cet exercice on considère le λ-calcul étendu aux paires. La paire des termes M et N est notée $\langle M, N\rangle$. Si $M : \tau_M$, lu « M est de type τ_N » et $N : \tau_N$, alors $\langle M, N\rangle : \tau_M \times \tau_N$. Les fonctions π_1 et π_2 sont les projections associées aux paires. Les projections se β-réduisant respectivement en $\pi_1\ \langle M, N\rangle \underset{\beta}{\rightarrow} M$ et $\pi_2\ \langle M, N\rangle \underset{\beta}{\rightarrow} N$.

1. Donner le type puis définir une fonction `c_add` qui prend un nombre en argument et renvoie une fonction qui à son tour prend un autre nombre en argument et renvoie la somme des deux.
2. Donner les types des projections π_1 et π_2.
3. Donner le type puis définir une fonction `curry2` qui prend en argument une fonction `f` à deux arguments et renvoie sa version curryfiée, c'est-à-dire la fonction qui prend le premier argument et renvoie une fonction qui prend le deuxième argument et renvoie le résultat de `f` appliquée aux deux arguments.
4. On définit `let plus = (x,y)=> x + y;`. Utiliser `curry2` pour proposer une nouvelle définition de `c_add`.
5. Soit la définition de `let combine = f => f(2)*f(3);`. Calculer `combine(c_add(5))`.
6. Simplifier en la β-réduisant l'expression `let f = x => combine(c_add(x));`.
7. Donner le type puis écrire une fonction `uncurry2` en JavaScript qui soit l'inverse de `curry2`, c'est-à-dire qui vérifie `uncurry2(curry2(f)) = f` et `curry2(uncurry2(f)) = f`.
8. Traduire les définitions de `uncurry2` et de `curry2` en λ-calcul (étendu aux paires). Ces termes seront nommés **uncurry** et **curry**.
9. On rappelle la définition de $\mathbf{K} = \lambda x.\lambda y.x$. Prouver que le curryfié de π_1 est $\mathbf{K}$. Quelle expression du λ-calcul correspond au curryfié de π_2.
10. Prouver les propriétés qui unissent `curry2` et `uncurry2` de la question 7 en β-réduisant $\mathbf{uncurry}(\mathbf{curry}(F))\langle X, Y\rangle$ et $\mathbf{curry}(\mathbf{uncurry}(F))\ X\ Y$ pour des termes F, X et Y donnés en paramètres.
11. Quel est l'intérêt, du point de vue de la performance, du résultat précédent ?

Exercice 2 : Curryfication appliquée à `map()`

On rappelle que la méthode `Array.prototype.map(f)` crée *un nouveau tableau* composé des images des éléments du tableau appelant par la fonction `f` donnée en argument. Pour un tableau `A = [v_0, v_1, ..., v_n]`, `A.map(f)` renvoi `[f(v_0), f(v_1), ..., f(v_n)]` et laisse `A` inchangé.

1. Donner le type puis écrire une fonction `c_map` qui prend en argument une fonction `f` et renvoie une fonction qui prend un tableau `t` et applique `f` à chaque élément de `t`, c'est-à-dire une version curryfiée de `Array.prototype.map(f)`.
2. Utiliser `c_add` et `c_map` pour définir une fonction qui ajoute 3 à tous les éléments d'un tableau sans utiliser `function` et `=>`.
3. Soit un tableau `t`. Expliquer ce que calcule la fonction `c_map(f => f(3))(t)`
4. Soit `A` un tableau qui contienne des objets représentant des personnes avec les champs *nom*, *prénom* et *age*. Définir une fonction qui transforme `A` en un tableau ne comportant que les ages. Le faire également en utilisant la fonction `let prj = c => o => o[c];`.