LIFAP5 – Programmation fonctionnelle pour le WEB
TD2 – programmation fonctionnelle en `javascript`

Licence informatique UCBL – Printemps 2017–2018

Exercice 1 : Curryfication et application partielle

1. Donner le type puis écrire une fonction `c_add` qui prend un nombre en argument et renvoie une fonction qui prend un autre nombre en argument et renvoie la somme des deux.
2. Donner le type puis écrire une fonction `curry2` qui prend en argument une fonction `f` à deux arguments et renvoie sa version curryfiée, c'est-à-dire une fonction qui prend le premier argument et renvoie une fonction qui prend le deuxième argument qui renvoie le résultat de `f` appliquée aux deux arguments.
3. On définit `let plus = (x,y)=> x + y;`. Utiliser `curry2` pour proposer une nouvelle définition de `c_add`.
4. Soit la définition de `let combine = f => f(2)* f(3);`. Calculer `combine(c_add(5))`.
5. Donner le type puis écrire une fonction `uncurry2` qui soit l'inverse de `curry2`, c'est-à-dire qui a les propriétés `uncurry2(curry2(f)) = f` et `curry2(uncurry2(f)) = f`.
6. Traduire les définitions de `uncurry2` et de `curry2` en λ-calcul
7. 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.
8. Prouver les propriétés qui unissent `curry2` et `uncurry2`.

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`.
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];`.

Corrections

Solution de l'exercice 1

Objectif : comprendre ce qu'est une fonction curryfiée et pourquoi on peut vouloir écrire de telles fonctions ; montrer que le λ-calcul permet de raisonner sur les programmes purs.

1. Le type est `c_add` : `number` $\to$ `number` $\to$ `number`.

```
let c_add = (n1 => n2 => n1 + n2);
```

2. Autrement dit, on souhaite que `curry2(f)(x)(y) = f(x,y)` et tout est dit. Le type est `curry2` : $(\alpha \times \beta \to \gamma) \to \alpha \to \beta \to \gamma$.

```
let curry2 = f => a => b => f(a,b);
```

3. Simplement, on curryfie `plus`

```
let c_add2 = curry2(plus);
```

4. On va montre qu'on peut facilement faire des applications partielles une fois une fonction curryfiée.

```
   combine(c_add(5))
= (f => f(2)* f(3))(c_add(5))
= c_add(5)(2)* c_add(5)(3)
= (5 + 2) * (5 + 3)
= 7 * 8
= 56
```

5. Le type est `uncurry2` : $(\alpha \to \beta \to \gamma) \to (\alpha \times \beta \to \gamma)$.

```
let uncurry2 = f => (x,y) => f(x)(y);
```

6. On définit **curry** $= \lambda f.\lambda x.\lambda y.f\langle x, y\rangle$ et **uncurry** $= \lambda f.\lambda p.f(\pi_1 p)(\pi_2 p)$ avec π_i les projections de la paire.

7. $$\begin{aligned}(\textbf{curry}\ \pi_1) &\overset{def}{=} (\lambda f.\lambda x.\lambda y.f\langle x, y\rangle)\pi_1 \\ &\overset{f}{\rightsquigarrow} \lambda x.\lambda y.\pi_1\langle x, y\rangle \\ &\overset{\pi_1}{\rightsquigarrow} \lambda x.\lambda y.x \\ &\overset{def}{=} \mathbf{K}\end{aligned}$$

   Le curryfié de π_2 est $\lambda x.\lambda y.y$

8. Pour la preuve on β-réduit simplement les expressions et on utiliser les règles $\pi_1\langle X, Y\rangle = X$ et $\pi_2\langle X, Y\rangle = Y$ On conclut par η-équivalence que **uncurry**(**curry**$(F)) = F$.

$$\begin{aligned}\textbf{uncurry}(\textbf{curry}(F))\langle X, Y\rangle &\overset{def}{=} (\lambda f.\lambda p.f(\pi_1 p)(\pi_2 p))((\lambda f.\lambda x.\lambda y.f\langle x, y\rangle)F)\langle X, Y\rangle \\ &\overset{f}{\rightsquigarrow} (\lambda f.\lambda p.f(\pi_1 p)(\pi_2 p))(\lambda x.\lambda y.F\langle x, y\rangle)\langle X, Y\rangle \\ &\overset{f}{\rightsquigarrow} (\lambda p.(\lambda x.\lambda y.F\langle x, y\rangle)(\pi_1 p)(\pi_2 p))\langle X, Y\rangle \\ &\overset{p}{\rightsquigarrow} (\lambda x.\lambda y.F\langle x, y\rangle)(\pi_1\langle x, y\rangle)(\pi_2\langle X, Y\rangle) \\ &\overset{x,y}{\rightsquigarrow} F\langle(\pi_1\langle X, Y\rangle), (\pi_2\langle X, Y\rangle)\rangle \\ &\overset{\pi_1}{\rightsquigarrow} F\langle X, (\pi_2\langle X, Y\rangle)\rangle \\ &\overset{\pi_2}{\rightsquigarrow} F\langle X, Y\rangle\end{aligned}$$

Pour l'autre sens, on conclut par deux η-équivalences que $\textbf{curry}(\textbf{uncurry}(F)) = F$.

$$\begin{aligned}
\textbf{curry}(\textbf{uncurry}(F))\ X\ Y &\overset{def}{=} (\lambda f.\lambda x.\lambda y.f\langle x,y\rangle)((\lambda f.\lambda p.f(\pi_1 p)(\pi_2 p))F)\ X\ Y \\
&\overset{f}{\rightsquigarrow} (\lambda f.\lambda x.\lambda y.f\langle x,y\rangle)(\lambda p.F(\pi_1 p)(\pi_2 p))\ X\ Y \\
&\overset{f}{\rightsquigarrow} (\lambda x.\lambda y.(\lambda p.F(\pi_1 p)(\pi_2 p))\langle x,y\rangle)\ X\ Y \\
&\overset{x,y}{\rightsquigarrow} (\lambda p.F(\pi_1 p)(\pi_2 p))\langle X,Y\rangle \\
&\overset{p}{\rightsquigarrow} F(\pi_1\langle X,Y\rangle)(\pi_2\langle X,Y\rangle) \\
&\overset{\pi_1,\pi_2}{\rightsquigarrow} F(X)(Y)
\end{aligned}$$

On peut ainsi conclure sur l'isomorphisme entre $X \times Y \to Z$ et $X \to (Y \to Z)$

Solution de l'exercice 2

1. Le type est `c_map` : $(\alpha \to \beta) \to \texttt{Array}(\alpha) \to \texttt{Array}(\beta)$. Il suffit d'utiliser la méthode `map` des tableaux pour l'implémentation :

```
1 let c_map = (f => t => t.map(f)) ;
```

2. Deux application partielles de fonctions :

```
1 let add3 = c_map(c_add(3));
```

3. Cette fonction prend un tableau qui contient des fonctions et applique chacune à la valeur 3. Par exemple, avec `t = [x => x+1, x => x*2, x => x*x]` on obtient `[ 4, 6, 9 ]`.
4. On définit directement `A.map( o => o.age)` ou `A.map(prj("age"))`.