RICM2 - 2004/05
Langage et Programmation 2

Programmation fonctionnelle et λ-calcul

Manipulations de fonctions

Les fonctions d'ordre supérieur :

Dans les langages fonctionnels, les fonctions peuvent être utilisées dans la définition d'autres fonctions. On appelle donc fonction d'ordre supérieur, une fonction qui prend comme arguments une ou plusieurs autres fonctions. Le résultat d'une fonction d'ordre supérieur est aussi une fonction.
Par exemple, la fonction qui prend en argument deux fonctions et qui retourne la somme de celles-ci s'écrit de la façon suivante en CaML :

$$\textit{let somme\_fonctions} = \textit{fun } f \rightarrow \textit{fun } \ g \rightarrow (\textit{fun } \ x \rightarrow f\ x + g\ x)$$
$$\textit{let somme\_fonctions } f\ g = (\textit{fun } \ x \rightarrow f\ x + g\ x)$$

Exercice 1
Donner la définition en CaML de la fonction qui rend la composée de deux fonctions.

Exercice 2
Donner la définition en CaML de la fonction qui calcule la drivée d'une fonction.

Programmation fonctionnelle et lambda calcul :

Les langages fonctionnels sont des langages dans lesquels la notion de fonction est centrale. La sémantique des langages fonctionnels, comme CaML, est directement basée sur les concepts du λ-calcul. Le λ-calcul (inventé par A. Church en1930) fournit une notation pour transformer une expression en une fonction :

$$f = \lambda x.u$$

On dit que la variable x est liée et que l'expression u est la portée de cette liaison. Un exemple de fonction simple en λ-calcul :

$$f = \lambda x.x + 1$$

Cette fonction peut s'écrire, en CaML, des deux façons suivantes :

$$\textit{let } f = \textit{fun } \ x \rightarrow x + 1 \qquad \text{ou} \qquad \textit{let } f\ x = x + 1$$

Le typage de cette fonction est exprimé de la manire suivante :

$val\ f : int \rightarrow int = <fun>$

Construction des termes du λ-calcul :

1. une variable x est un λ-terme.
2. abstraction : si x est une variable, et u un λ-terme, alors $\lambda x.u$ est un λ-terme.
3. application : si u et v sont des λ-termes, alors (uv) est λ-terme.

Propriétés du λ-calcul : Convention de simplification d'écriture :

1. Associativité à gauche : $(((uv)(wt)s)$ s'abrège en $uv(wt)s$
2. Abstraction associative à droite : $\lambda x.(\lambda y.(\lambda z.u))$ s'abrège en $\lambda x.\lambda y.\lambda z.u$
3. Regroupement des λ : $\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.u))$ s'abrège en $\lambda x_1 x_2 \ldots x_n.u$

α-conversion : renommer toutes les occurrences d'une variable liée dans la portée de sa liaison.

Ex : $\lambda y.((xz)(\lambda x.xy))$, x renommé en s : $\lambda y.((xz)(\lambda s.sy))$

On s'autorise l'utilisation des fonctions prédéfinies usuelles sur les entiers.

β-réduction : donner une sémantique aux λ-termes qui les interprètent comme une application.

On note le résultat : $(\lambda x.u)v \rightarrow u[v/x]$

Ex : $(\lambda x.x + 1)(a + b) \rightarrow (a + b) + 1$

Les entiers de Church :

On veut représenter les entiers naturels dans le λ-calcul. Il existe plusieurs codages, mais le plus simple et le plus connu reste celui qui utilise les entiers de Church. Les entiers de Church peuvent être définis comme des opérateurs sur les fonctions. Par exemple, 3 est l'opérateur $\lambda f.\lambda x.f(f(fx))$. L'entier de Church est donc la fonctionnelle qui prend une fonction f et un argument x, et qui retourne f composée n fois appliquée à x.

Exercice 3

Écrire, en λ-calcul et en CaML, les fonctions de définition des entiers de Church (*zéro*, *un*, *deux*, puis *successeur*). Donner le typage de chaque fonction. Réduire (*successeur* 2).

Exercice 4

Écrire, en λ-calcul et en CaML, les fonctions qui définissent les opérations (*somme*, *produit* et *exponentielle*) appliquées aux entiers de Church. Donner le typage de chaque fonction.

Exercice 5
Écrire, en CaML, les deux fonctions de conversions entre entiers de Church et entiers naturels.

Logique :

On veut représenter les valeurs de logique grâce au λ-calcul.

Exercice 6
Donner, en λ-calcul et CaML, la représentation des valeurs vrai et faux par des fonctions. Donner le typage de chaque fonction.

Exercice 7
Donner en λ-calcul et CaML l'implémentation des opérations associées à la logique (*and*, *or* et *not*). Donner le typage de chaque fonction.