TD 11 – λ-calcul

Rappels

Le λ-calcul est un système formel (purement syntaxique) introduit par A. Church dans les années 1930. Il constitue le cœur des langages de programmation fonctionnels contemporains comme OCaml ou Haskell. Les termes (éléments syntaxiques) du λ-calcul sont appelés *λ-termes* et suivent la syntaxe récursive suivante (où x représente une *variable* et M un terme) :

$$M ::= x \mid M\,M' \mid \lambda x.M$$

Ainsi, un λ-terme est soit une variable x (penser à une variable `x` en OCaml), soit l'*application* $M\,M'$ d'un terme M' à un terme M (penser à l'application `M M'` d'un fonction `M` à un argument `M'` en OCaml), soit une fonction qui prend un paramètre représenté par la variable x et dont le corps est le terme M (penser à `fun x -> M` en OCaml).

On associe à cette syntaxe un système de réécriture, noté $\to$, dont la seule règle[1] est la *β-réduction* (où $M[M'/x]$ désigne le terme M dans lequel on a remplacé toutes les occurences de la variable x par le terme M') :

$$(\lambda x.M)M' \longrightarrow M[M'/x]$$

Ainsi l'application d'une fonction $\lambda x.M$ à un aurgument M' se réécrit en le corps $M[M'/x]$ de la fonction dans lequel on a remplacé les x par des M. On note $\to^*$ la clôture transitive de la β-réduction.
Il existe plusieurs façons de représenter les entiers naturels par des λ-termes. Nous considérerons ici les *entiers de Church* définis par :

$$\overline{n} = \lambda f.\lambda x.(\underbrace{f\,(f\,(\ldots(f}_{n}\,x)\ldots)) = \lambda f.\lambda x.(f^n\,x)$$

On dira qu'une fonction $f : \mathbb{N}^p \to \mathbb{N}$ est *λ-représentable* s'il existe un λ-terme F tel que pour tout $(n_1, \ldots, n_p) \in \mathbb{N}^p$, on a : $F\overline{n_1}\ldots\overline{n_p} \to^* \overline{f(n_1, \ldots, n_p)}$.

Exercice 1. *Pas si beta*

1. Soit $I = \lambda x.x$ et $L = \lambda yx.y(y(I))$. Réduire $L(II)$.
2. Soit $I = \lambda x.x$ et $F = \lambda xy.(y(yI))$. Réduire $F(II)F(II)$.
3. Réduire $\Omega = (\lambda x.xx)(\lambda x.xx)$. Proposer un λ-terme dont la taille augmente à l'infini au fil des réductions.
4. Proposer des λ-termes permettant de modéliser la notion de couple, ainsi que les deux projections associées.
5. Donner des λ-termes pour les booléens.

1. Les autres règles de réduction du λ-calcul, que l'on n'énoncera par formellement ici, ne servent qu'à permettre l'usage de la β-réduction à l'intérieur d'un terme.

6. En remarquant qu'un entier peut être vu comme une imbrication de paires, donner une λ-représentation des entiers, différente de celle de Church.

Exercice 2. *Une idée fixe*

1. Donner des λ-termes correspondant aux combinateurs de base : $K : (f, g) \mapsto f$, identité $I : f \mapsto f$, composition $B : (f, g) \mapsto f \circ g$, commmutation $C : f \mapsto f^c$ où $f^c(x, y) = f(y, x)$ et diagonalisation $W : f \mapsto f^\delta$ où $f^\delta(x) = f(x, x)$.
2. Montrer que tout λ-terme admet un point fixe : pour tout λ-terme a, il existe un λ-terme X tel que $X \to^* aX$.
3. Un *combinateur de point-fixe* est un λ-terme F tel que pour tout a, on a $Fa \to^* a(Fa)$.
   Donner un exemple de combinateur de point-fixe.

Exercice 3. *Encore à faire des + et des - en L3...*

1. Montrer que les fonctions récursives de base (constantes, projections et successeur) sont λ-représentables.
2. Montrer que la fonction addition est λ-représentable.
3. Montrer que la fonction prédécesseur est λ-représentable.
   Indice : utiliser le λ-terme C de couple et encoder la fonction $S : (m, n) \to (n, n + 1)$.
4. Donner des λ-termes représentant eq_0 (fonction de test en 0), la multiplication et l'exponentiation.

Exercice 4. *Clôturons le semestre*

Montrer que l'ensemble des fonctions λ-représentables est clos par :

1. composition.
2. schéma de récurrence (*Indice : commencer par montrer la clôture par schéma d'itération.*).
3. schéma de minimisation.

Exercice 5. *Spécial dédicace à D.*

1. Expliquer pourquoi tout λ-terme est calculable par une machine de Turing.
2. À l'aide des exercices précédents, conclure sur l'expressivité (en terme de calculabilité) du λ-calcul.
3. Donner une idée de comment simuler de manière directe une exécution de machine de Turing par un λ-terme.
   Indice : on pourra redéfinir en λ-calcul les primitives d'un langage simple de programmation, ainsi que des structures de données comme des listes chaînées.