Cours de logique
TD 3

Lambda-calcul, preuves et programmes

Polytech Marseille - 3A
Filière Informatique

Séance 1

Nous allons dans cette séance nous intéresser au λ-calcul pur vu comme langage de programmation (et à son expressivité).

Exercice 1 (Codage des booléens)**.** On rappelle le codage des booléens vu en cours :

$$\begin{aligned} \top &= \lambda x.\lambda y.x \\ \bot &= \lambda x.\lambda y.x \\ \text{if} - \text{then} - \text{else} &= \lambda p.\lambda a.\lambda b.(p)\ a\ b \end{aligned}$$

(i) Montrez que $(\text{if} - \text{then} - \text{else})\ \top\ a_0\ b_0 \to_\beta^* a_0$ et $((\text{if} - \text{then} - \text{else})\ \bot\ a_0\ b_0 \to_\beta^* b_0$.

(ii) Pourquoi est-ce bien le comportement attendu ?

(iii) Ecrire des λ-termes codant la négation (NOT), la conjonction (ET) et la disjonction (OU). Comme pour tout programme, différentes solutions existent. Vous essaierez de trouver les termes les plus compacts possibles.

Exercice 2 (Codage des couples)**.** L'encodage des coupes se fait au moyen de 3 fonctions : l'une constituant un couple à partir de deux arguments ($\langle\cdot,\cdot\rangle$), et deux fonctions de projection renvoyant le premier et second élément du coupe :

$$\begin{aligned} \langle\cdot,\cdot\rangle &= \lambda a.\lambda b.\lambda p.(p)\ a\ b \\ \pi_1 &= \lambda p.(p)\ \top \\ \pi_2 &= \lambda p.(p)\ \bot \end{aligned}$$

(i) A quelle opération booléenne l'opération $\langle\cdot,\cdot\rangle$ ressemble-t-elle ?

(ii) Prouvez que la composition de ces fonctions correspond bien à un comportement de couple.

Exercice 3 (Codage des entiers)**.** Tout $n \in \mathbb{N}$, est codé par le λ-terme :

$$\bar{n} = \lambda f.\lambda x.\underbrace{(f)\ (f)\ \ldots (f)}_{n \text{ fois, noté } f^{(n)}} x$$

On donne alors les fonctions suivantes :

$$\begin{aligned}
\text{incr} &= \lambda n.\lambda f\lambda x.(f)\ (n)\ f\ x \\
\text{egal0} &= \lambda n.(n)\ (\lambda b.\bot)\ \top \\
\text{add} &= \lambda n.\lambda m.((m)\ incr)\ n \\
\text{mult} &= \lambda n.\lambda m.(((m)\ \text{add})\ n)\ \bar{0} \\
\text{iter} &= \lambda n.\lambda f.\lambda x_0.(n)\ f\ x_0
\end{aligned}$$

(i) Montrez que :

$$(\text{incr})\ \bar{n} \to_\beta^* \overline{n+1}$$

(ii) Montrez que :

$$(\text{egal0})\ \bar{0} \to_\beta^* \top \quad \text{et} \quad \text{egal0}\ \bar{n} \to_\beta^* \bot\ (\text{pour } n > 0)$$

(iii) Montrez par récurrence que :

$$(\text{add})\ \bar{n}\ \bar{m} \to_\beta^* \overline{n+m}$$

(iv) Montrez que iter est un itérateur sur les entiers, donc :

$$(\text{iter})\ \bar{n}\ f\ x_0 \to_\beta^* f^n(x_0)$$

(v) On définit alors le terme :

$$\text{decr} = \lambda n.(\pi_1)\,((iter)\ n\ (\lambda c.\langle(\pi_2)c, (\text{incr})(\pi_2)c\rangle)\ \langle 0,0\rangle)$$

A votre avis, que calcule-t-il et comment ?

(vi) (Opt) Montrez par récurrence que :

$$\text{mult}\ \bar{n}\ \bar{m} \to_\beta^* \overline{n \times m}$$

Exercice 4 (Récursivité)**.** Pour qu'un langage soit Turing-complet, il faut qu'il puisse itérer du code ; et en l'occurrence, quand on manipule des fonctions, il faut que le langage permette la récursivité.

On considère le λ-terme suivant :

$$\Theta = \left(\underbrace{\lambda z.\lambda x.(x)\ ((z)\ z\ x)}_{Z}\right) \underbrace{\lambda z.\lambda x.(x)\ ((z)\ z\ x)}_{Z}$$

(i) Montrez que $\Theta \to_\beta \lambda x.(x)((Z)Zx) = \lambda x.(x)((\Theta)x)$.

(ii) En déduire que pour tout λ-terme f : $(\Theta)f \to_\beta^* (f)((\Theta)f)$.

(iii) Enfin, si :

$$t = \lambda f.\lambda n.((\text{if} - \text{then} - \text{else})\ ((\text{egal0})\ n)\ \bar{1}\ ((\text{mult})\ n\ ((f)\ (\text{decr})\ n))$$

puis :

$$\text{fun} = (\Theta)\ t$$

que calculent (fun) $\bar{1}$ ou (fun) $\bar{3}$?

Séance 2

Nous allons dans cette séance nous intéresser au typage du lambda-calcul et à l'isomorphisme de Curry-Howard (donc l'équivalence preuves/programmes).

Nous nous placerons dans le cadre d'un système de typage du second ordre (donc intégrant les connecteurs de la logique propositionnelle, mais aussi le quantificateur $\forall$ du second ordre (donc sur les variables propositionnelles). Les types (ie. les formules) sont donc définis de la manière suivante :

$$\begin{array}{lll} \text{type} & ::= X, Y, \ldots & \text{(variables propositionnelles)} \\ & \mid \tau \Rightarrow \sigma & \\ & \mid \forall X.\tau & \text{(quantificateur du second ordre)} \end{array}$$

On considère alors les ensembles suivants de règles de typage (et leurs équivalents comme règles de déduction naturelle) :

Groupe structurel

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma}\ \text{var}$$

Groupe logique

$$\frac{\Gamma, x : \sigma \vdash t : \tau}{\Gamma \vdash \lambda x.t : \sigma \Rightarrow \tau}\ \Rightarrow_{\text{intro}}$$

$$\frac{\Gamma \vdash t_1 : \sigma \Rightarrow \tau \qquad \Gamma \vdash t_2 : \sigma}{\Gamma \vdash (t_1)t_2 : \tau}\ \Rightarrow_{\text{élim}}$$

$$\frac{\Gamma \vdash t : \sigma \quad X \notin \Gamma}{\Gamma \vdash t : \forall X.\sigma}\ \forall_{\text{intro}}$$

$$\frac{\Gamma \vdash t : \forall X.\sigma}{\Gamma \vdash t : \sigma[\tau/X]}\ \forall_{\text{élim}}$$

Exercice 5. Dans cet exercice on s'intéresse aux entiers de Peano.

$$\bar{n} = \lambda f.\lambda x.\underbrace{(f)\ (f)\ \ldots (f)}_{n \text{ fois, noté } f^{(n)}} x$$

On rappelle la définition des fonctions classiques sur les entiers :

$$\begin{array}{rcl} \text{incr} & = & \lambda n.\lambda f \lambda x.(f)\ (n)\ f\ x \\ \text{egal0} & = & \lambda n.(n)\ (\lambda b.\bot)\ \top \\ \text{add} & = & \lambda n.\lambda m.((m)\ incr)\ n \\ \text{mult} & = & \lambda n.\lambda m.(((m)\ \text{add})\ n)\ \bar{0} \\ \text{iter} & = & \lambda n.\lambda f.\lambda x_0.(n)\ f\ x_0 \end{array}$$

(i) Montrez que $\bar{n} : \forall X.((X \Rightarrow X) \Rightarrow X \Rightarrow X$. On appellera $\mathbb{N}$ ce type.

(ii) Montrez que $\text{incr} : \mathbb{N} \Rightarrow \mathbb{N}$.

(iii) En déduire que $\text{add} : \mathbb{N} \Rightarrow \mathbb{N} \Rightarrow \mathbb{N}$.

On associe ainsi des preuves aux lambda-termes typables ; les fonctions codées par ces termes fournissent une "version calculatoire" des preuves (produisant leur conclusion à partir des hypothèses). C'est l'un des sens de Curry-Howard.

Exercice 6. On s'intéresse ensuite à l'autre sens de l'isomorphisme de Curry-Howard. Considérons les trois preuves suivantes de la déduction naturelle :

$$\dfrac{\dfrac{\dfrac{}{X \Rightarrow X, X \vdash X}\,\text{var}}{\vdash (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\Rightarrow_{\text{intro}}, \Rightarrow_{\text{intro}}}{\vdash \forall X.((X \Rightarrow X) \Rightarrow X \Rightarrow X)}\,\forall_{\text{intro}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X \Rightarrow X, X \vdash X \Rightarrow X}\,\text{var} \quad \dfrac{}{X \Rightarrow X, X \vdash X}\,\text{var}}{X \Rightarrow X, X \vdash X}\,\Rightarrow_{\text{élim}}}{\vdash (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\Rightarrow_{\text{intro}}, \Rightarrow_{\text{intro}}}{\vdash \forall X.((X \Rightarrow X) \Rightarrow X \Rightarrow X)}\,\forall_{\text{intro}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X \Rightarrow X, X \vdash X \Rightarrow X}\,\text{var} \quad \dfrac{\dfrac{}{X \Rightarrow X, X \vdash X \Rightarrow X}\,\text{var} \quad \dfrac{}{X \Rightarrow X, X \vdash X}\,\text{var}}{X \Rightarrow X, X \vdash X}\,\Rightarrow_{\text{élim}}}{X \Rightarrow X, X \vdash X}\,\Rightarrow_{\text{élim}}}{\vdash (X \Rightarrow X) \Rightarrow X \Rightarrow X}\,\Rightarrow_{\text{intro}}, \Rightarrow_{\text{intro}}}{\vdash \forall X.((X \Rightarrow X) \Rightarrow X \Rightarrow X)}\,\forall_{\text{intro}}$$

Quels sont les lambda-termes associés ? Donc que "calculent" ces preuves ?

Exercice 7. On peut généraliser la construction des entiers à n'importe quel type inductif. Etant donné un type truc(X) défini inductivement :

$$\begin{array}{rcl} \text{truc}(X) & ::= & \text{Cas}_1(X, \text{truc}(X)) \\ & | & \text{Cas}_2(\text{int}, X) \\ & | & \text{Cas}_3 \end{array}$$

Une donnée de ce type est un opérateur qui attend une fonction pour chaque cas et construit la donnée avec ces fonctions :

$$\begin{array}{lcl} \text{Cas}_1(a,t) & = & \lambda c_1.\lambda c_2.\lambda c_3.(c_1)\ a\ ((t)\ c_1\ c_2\ c_3) \\ \text{Cas}_2(n,a) & = & \lambda c_1.\lambda c_2.\lambda c_3(c_2)\ n\ a \\ \text{Cas}_3 & = & \lambda c_1.\lambda c_2.\lambda c_3.c_3 \end{array}$$

(i) Décrire le type Liste comme un type inductif (on baptisera "Cons" et "Empty" les deux cas)

(ii) En déduire un codage des listes (vous ferez apparaître les fonctions du type abstrait de données Listes vues en cours d'algorithmique : `creer_liste_vide` et `ajouter`).

(iii) Obtenez-vous des termes typables ? Quel est le type d'une liste ?

(iv) Est-il facile, dans le λ-calcul pur, de définir les autres fonctions du TAD (`tete`, `queue`, `est_liste_vide` ?)

(v) En revanche, écrivez une fonction calculant la somme des éléments stockés dans une liste.