SORBONNE UNIVERSITÉ
CRÉATEURS DE FUTURS DEPUIS 1257

TAS TD 2 - Lambda-Calcul Typé

On rappelle les règles de typage *à la Curry* (c-à-d, sans que les types apparaissent explicitement dans les termes) du λ-calcul simplement typé :

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \qquad \frac{\Gamma \vdash M : \sigma \to \tau \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash M\ N : \tau}$$

1 Dérivations de typage

1.1 Typages simples

On (re)définit les termes suivants :

$$\mathbf{I} = \lambda x.x \qquad \mathbf{K} = \lambda xy.x \qquad \mathbf{S} = \lambda xyz.(xz)\ (yz)$$

1. Typer $\mathbf{I}$, $\mathbf{K}$ et $\mathbf{S}$. Puis typer $(\mathbf{I}\ \mathbf{I}\ (\mathbf{I}\ \mathbf{I}))$.
2. Typer $\mathbf{SKK}$.
3. Pouvait-on déduire le type de $\mathbf{SKK}$ sans faire la dérivation ?
4. Trouver un habitant $\mathbf{B}$ du type $(\tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho$
5. Typer le terme $\lambda xyzt.x\ (y\ t)\ t\ (z\ t)$
6. Prouver que $\omega = \lambda x.x\ x$ et $\lambda ab.(a\ b)\ (b\ a)$ ne sont pas typables.

1.2 Types non- habités

Soit E un type sans habitant.

1. Dire lesquels des types suivants sont habités par des termes clos :

$$\sigma \to E \qquad E \to E \qquad \sigma \to ((\sigma \to E) \to E) \qquad ((\sigma \to E) \to E) \to \sigma \qquad (\sigma \to E) \to \sigma \to E$$
$$((\sigma \to \tau) \to \sigma) \to \sigma$$

1.3 Modélisation

Une des définition des entiers de Church est $[\![n]\!] = \lambda xf.f^n\ x$

Un des encodages de la *liste* `[e1, e2, ..., ek]` est $\lambda cn.c\ e_1\ (c\ e_2\ \dots (c\ e_k\ n) \dots)$

1. Donner le type des entiers de Church.
2. Donner une dérivation de typage ou une preuve de non-typabilité de successeur, addition, multiplication et puissance.
3. Donner le type des listes.
4. Donner une dérivation de typage ou une preuve de non-typabilité de tête et *map*.

2 Transformations CPS

La programmation par continuation (*Continuation Passing Style*) est un paradigme permettant de rendre explicite le flot de calcul dans les programmes, ce qui permet des transformations utiles par exemple pour la compilation, ou pour encoder un mécanisme d'exceptions.

Comme dans les TPs précédents, les termes écrits en CPS attendent un futur (une continuation k) qui représente la suite du calcul.

On (re)définit le λ-calcul avec la grammaire suivante :

$$\begin{array}{lrcl} \text{Valeurs} & U, V & ::= & x \mid \lambda x.M \\ \text{Termes} & M, N & ::= & V \mid MN \end{array}$$

On définit alors deux traductions (mutuellement récursives) du λ-calcul dans lui-mÃªme par :

$$\begin{array}{rclcrcl} [x] & = & x & & [\![V]\!] & = & \lambda k.k[V] \\ [\lambda x.M] & = & \lambda x.[\![M]\!] & & [\![MN]\!] & = & \lambda k.[\![M]\!](\lambda\alpha.[\![N]\!](\lambda\beta.\alpha\beta k)) \end{array}$$

2.1 Encodage

1. Ecrire $[\![\omega I]\!]$ et le réduire en lui donnant la continuation initiale I.
2. Montrez que, si V est une valeur, $[\![M\{V/x\}]\!] = [\![M]\!]\{[V]/x\}$.

2.2 Etapes intermédiaires

La fonction $|M|_k$ décrit les étapes de *réductions administratives* lors de l'application de la traduction de M à une continuation k :

$$\begin{array}{rclcrcl} |V|_k & = & k[V] & & |VN|_k & = & |N|_{\lambda\beta.[V]\beta k} \\ |UV|_k & = & [U][V]k & & |MN|_k & = & |M|_{\lambda\alpha.[\![N]\!](\lambda\beta.\alpha\beta k)} \end{array}$$

1. Prouvez que, pour tout λ-terme K, $[\![M]\!]K \to_v^+ |M|_K$.
2. Le résultat précédent est-il vrai pour $\to_n$?
3. Démontrez que, si $M \to_v N$, alors pour tout K, $|M|_k \to_v^+ |N|_k$. Et pour $\to_n$?
4. Prouvez l'*indifférence* de la transformation : $[\![M]\!]I \to_v^* V \iff [\![M]\!]I \to_n^* V$.
5. Démontrez le théorème de *simulation* : si $M \to_v^* V$, alors $[\![M]\!]I \to^* [V]$.

2.3 Typage

1. Trouver la traduction sur les types permettant de prouver :
   Si $\Gamma \vdash T : \theta$ alors $[\![\Gamma]\!] \vdash [\![M]\!] : [\![\theta]\!]$.
   (Par $[\![\Gamma]\!]$ on entend $\{(x : [\![\tau]\!])$ tels que $(x : \tau) \in \Gamma\}$)
2. Que peut-on en déduire sur le comportement de la traduction CPS vis-à-vis de la terminaison ?

3 Système F

Voici la syntaxe de Système F *à la Church* (c'est-à-dire avec la présence explicite de type dans les termes) :

$$M ::= x \mid (M\ M) \mid \lambda x^T.M \mid \Lambda X.M \mid (M\ T) \qquad\qquad T ::= o \mid T \to T \mid X \mid \forall X.T$$

la sémantique :

$$\frac{}{(\lambda x^\sigma.M)\ N \to M[N/x]} \qquad \frac{}{(\Lambda X.M)\ T \to M[T/X]} \qquad (\xi) \qquad (\mu_l) \qquad (\mu_r)$$

et les règles de typage :

$$\frac{}{\Gamma, x : \sigma \vdash x : \sigma} \qquad \frac{\Gamma, x : \sigma \vdash T : \tau}{\Gamma \vdash \lambda x^{\sigma}.T : \sigma \to \tau} \qquad \frac{\Gamma \vdash T : \sigma \to \tau \qquad \Gamma \vdash U : \sigma}{\Gamma \vdash T\ U : \tau}$$

$$\frac{\Gamma; X \vdash T : \sigma}{\Gamma \vdash \Lambda X.T : \forall X.\sigma} \qquad \frac{\Gamma \vdash T : \forall X.\sigma}{\Gamma \vdash T\ \tau : \sigma[\tau/X]}$$

3.1 Manipulation

1. Donner un type (où une écriture *à la Church*) et une dérivation de typage de ces termes écrits *à la Curry* :

   $\lambda x.x$ $\qquad$ $\lambda x.(x\ x)$ $\qquad$ $\lambda x.\lambda y.x\ (y\ x)$ $\qquad$ $\lambda f.((f\ \mathbf{I})\ (f\ y))$

   $\mathbf{double} = \Lambda X.\lambda f.\lambda x.f(fx)$ $\qquad$ $\mathbf{quadruple}$

2. Donner des habitants des types suivants :

   $\forall X_1.\forall X_2.(X_1 \to X_2 \to X_1)$ $\qquad$ $(\forall X_1.X_1 \to X_3) \to (\forall X_2.X_2) \to X_3$ $\qquad$ $\forall X.X$

3. Décrire l'ensemble des habitants du type $\forall X.X \to X$.

3.2 Entiers

1. Etendre la définition des entiers de Church avec la puissance du polymorphimse.
2. Coder $\mathbf{power}$.

3.3 Let vs. F

1. Expliquer les différences entre le let-polymorphisme vu en cours et celui de Système F.